A STUDY OF MODULE TEAM ORGANISATION IN INDIAN SOFTWARE INDUSTRY

A Thesis Submitted

in Partial Fulfillment of the Requirements

for the Degree of

Master of Technology

by

DEBASISHA MISHRA

to the

DEPARTMENT OF INDUSTRIAL AND MANAGEMENT ENGINEERING

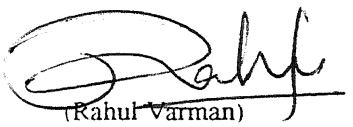
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

APRIL 1997

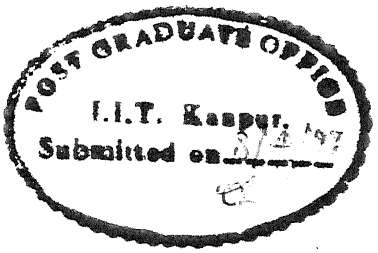
CERTIFICATE

It is to certify that the work contained in the thesis entitled "**A STUDY OF MODULE TEAM ORGANISATION IN INDIAN SOFTWARE INDUSTRY**" by Debasisha Mishra (Roll No. 9511402) has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

April, 1997.



(Rahul Varman)
Asst. Professor.
Industrial and Management Engineering.
Indian Institute of Technology.
Kanpur - 208016.



IME-1997-M-MIS-MOD

ABSTRACT

The Indian software industry has been one of the better performing industries in India. Many of the Fortune 500 companies prefer to outsource their software requirement from India. Despite such superlative performances, the industry has been unable to take big projects.

The broad research objective of this study is to find out the team organisations for various types of software teams differentiated according to type of software Application or System, and size of the team. The team is called Small module team where the number of people working is less than or equal to four, otherwise it is called a Large team. The team structure was compared with various ideal team structures proposed in the literature in order to test the validity of ideal team structures in the Indian context.

The questionnaire survey method was followed for this research work. The study encompassed twenty five companies situated at various Software Parks, Export Processing Zones and Industrial Zones of Noida and New Delhi. Since there are two broad categories of software professionals - System Analyst and Programmer, one response from each division of labour was collected from each organisation. In total forty-six responses were collected from twenty five companies.

The findings of the study show that the actual team structures resemble the ideal team structures, but with certain variations. The type of software has no influence on the team structure. Two types of team structures were found for Small teams with certain variations to that of ideal team structures of Egoless Programming team and Chief Programmer team. The Large team structures resembles to Chief Programmer team with some variations.

Reasons for the variations and implications of the ideal structures for the Indian software industry have been discussed.

ACKNOWLEDGMENT

I would like to express my heartfelt thanks to my thesis supervisor Dr. Rahul Varman. Despite the best of my resistance, he was able to lead me through a lucid, step by step thought process, which helped me drawing the conclusions. Whenever I committed mistake, he always corrected me and continued to enthuse and encourage me. I thank him for all his help, inspiration, encouragement - and making me realize various procedural aspects of research methodology.

I acknowledge my gratitude to Mr. Ashok Laha for his valuable comments on the questionnaire which helped it to get into the final shape. He gave me rare insights about the practical aspects of Indian software industry which inspired me in analysis of the data.

I thank all the members of the IME family for their direct and indirect contribution towards providing an atmosphere which helped in promoting my personal and professional growth. I owe my special thanks to my friends Ved and Pathak for providing me valuable suggestions at various stages of the thesis work.

I express my gratitude towards all the software professionals who spared their valuable time to fill the questionnaire without which this piece of research work would not have seen the light of the day. I thanks all the software professionals who helped me in making of the questionnaire.

I thank my friend Debansu and Ashutosh for providing me a pleasurable atmosphere for staying in Noida and extending me all possible help in the data collection process. Without their help I would have lost in the big metro city.

CONTENTS

LIST OF FIGURES

Chapter 1

1. INTRODUCTION

> "What do the following have in common : Commuters on the London underground; customer of Citibank, American Express, Deutsche Bank; Manufacturers of GE, IBM, Reebok, and GM; and the passengers on Swissair, American Airlines, and Singapore Airlines ? They all depend on computer software developed in India".
>
> (Business World, July 1995).

The above article is one of many which focuses on Indian Software Industry. There is growing importance of the software industry all over world. It is because Computer - a marvel of science and technology has found its way into every aspect of not only business, but also in the life of common man/woman. This complexity of the computer produced a crucial characteristic of computer industry. The computer industry has vertically disintegrated into two parts, namely software and hardware industry. The two industries are showing completely different characteristics . As the hardware industry is increasingly automated to reduce the cost of manufacturing, the software industry is still labour intensive. The use of tools are not prevalent to reduce the cost of making the software. As the time approaches the twenty-first century, the hardware cost is rapidly decreasing where as software cost is increasing.

The role of software has spread to production, engineering, education and general studies, etc. Starting from bank, hospitals, educational centres to power plant, automobile manufacturing industries to spacecraft industry are dependent on different kinds of softwares to improve their productivity and remain competitive in the cut throat competition of the current world.

Due to the need of software in almost every field of industry, the software industry is booming and growing faster than most other industries. The market for computer software and service is global in nature, intensely competitive, fast changing and fast growing. The global market, in 1996, for software is estimated to be any where between US $200 billion to US $300 billion. The industry has shown a growth rate of 20% and is estimated to continue at around 15% globally (Schware, 1992)

The need for software has increased multifold because the hardware sophistication has increased substantially outpacing the ability to build software that can tap hardware's potential. The ability to build new programs cannot keep pace with the demand for new programs. As the complexity and criticality of systems increases, there is greater demand for more reliable and easy to maintain software.

As the software is becoming a necessity in each and every field, the range of expertise and knowledge required for it's development has increased substantially. So project management in software industry has become a formidable, if not an impossible task (Pressman, 1989). The importance of human resource management has increased as software is still labour intensive and people form different field and back grounds have

to work together as a team. Managing software engineering projects require an ability to comprehend and balance technological, economic and social bases through which software systems are developed. It requires people who can formulate strategies for developing systems in presence of ill-defined requirements, new computing technologies and recurring dilemma with existing computing arrangements. This necessarily assumes skill in acquiring adequate computing resources, controlling projects, co-ordinating development schedules, employing and directing competent staff. It also requires people who can organise the process for developing and evolving software products with locally available resources.The members have to interact heavily among themseleves to convey their ideas to one another for smooth process of development. So managing the software engineering project is as much a job of social interaction as it is one of technical direction. This piece of research work addresses the social aspect of software development and maintenance.

1.1 *INDIAN SOFTWARE SCENARIO*

1.1.1 *Strengths*

Over the years, the Indian software industry has built an enviable global reputation for low cost and high quality. The spiralling revenues reflect this trend. Exports have grown from Rs. 50 Crores in 1985, Rs. 1,535 Crores in 1994-95, to Rs. 2,410 Crores in 1995-96. The Indian industry has grown at a rate that is double the world average. In 1994, the export's growth rate was 52%. In 1995-96, exports grew by 57%. In dollar terms, these revenues are at a figure of $725 million. This far surpasses the world bank predictions, made in 1992, of $660 million for 1996. The industry is a net earner of foreign exchange. The earnings have risen to 45% of the total software exports. A total

of more than 150 companies exported software worth Rs. 1 Crores each in 1996. There were only 10 companies in this category in 1990. In terms of markets, the USA accounted for more than 50 percent of the total exports. Exports to the Europe accounted for 22% (NASSCOM 1996).

On the quality front, India has been exceptional. Forty-one companies have qualified for ISO 9000 certification and many other are in the process of doing so. The Indian software industry would have the largest number of ISO 9000 certified companies in any industry, anywhere in the world (Business World, 1995).

So it is not surprising, that a recent World Bank survey of US companies rated India as the first preference for sourcing software and services. This, against seven competing countries- Israle, Ireland, Singapore, Philippines, China, Hungry and Mexico. Over 104 out of the Fortune 500 companies outsource to Indian companies. With such an established reputation, the Indians are now looking towards new markets in EEC, Australia, South Africa, Japan & the Asia-Pacific regions.

Considering such an encouraging performance, the Department of Electronics (DoE) has formed a committee to study the software industry and its export potential. This committee is part of DoE's 9^{th} five-year plan (1997- 2002). This committee intends to target Rs. 21,000 Crores by the year 2002. The chairperson of this committee said that there is lot of areas in which the industry has to improve (NASSCOM, 1996).

1.1.2 *Weakness*

Lack of Package Orientation - Although, a few companies have started making good software packages, the industry as a whole is still not oriented towards development of " Shrink Wrapped " software packages, and is thus not able to take advantage of multiplier effect of growth in revenues.

Original Technology - Indian software industry has good expertise to use latest technology . However, with few exceptions, it has still not produced enough original technology in the country. In other words, the industry has not created original operating systems or new computer languages.

Project Management Skills - Although many Indian companies have started taking large projects of more than 300 - 400 man years, the industry as a whole is not oriented towards developing large software products. Many companies run virtual organisation where they are only interested in body shopping. A large number of Indian companies are developing small parts of bigger overseas projects. It is a very low value added work mostly involving coding without doing deeper aspects of design and project management. It will have detrimental effect on the industry in the long run.

The present research tries to focus Project Management aspect of software development in the Indian industry.

1.2 *THE SCOPE OF THE THESIS*

This thesis is an attempt to make a contribution in the direction of Project Management. It is a study on Indian software industry. The sample for the study was a group of Indian software companies situated in New Delhi and Noida. The study focuses on the development of software module. It doesnot differentiate companies according to type of market they are catering, service or product they are providing to the customer etc. It doesnot differentiate the companies by technology they are using for development work. It confines itself to organisational aspects of the software development team.

1.3 *OVERVIEW*

The thesis is a study of software development team. It differentiates the modules according to the number of people working in it and type of software namely Application software and System software. The modules are termed as 'small' modules where the number of people working in it is equal to or less than four. Whenever the number of people working in a project are more than four it is termed as a 'large' module. The composition of team, the division of labour in it was found out from the responses given by the software companies to the questionnaire designed for the study.

The objectives of this research work is

1. To find out the actual working style of software module teams and compare it with different ideal structures proposed by different researchers (in USA).

2. To find out difference of team structure for different types of software, like Application software and System software.

1.4 *RELEVANCE*

The Indian software industry is too recent to have ready made business management models that could be taught and applied to improve the business. Applying the knowledge from Project Management literature can gives us a tested and known way to improve our current software project management.

Successfully applying the Project Management literature can give the software development manager readily available are tested model that can give immense insight into how to better manage people in the development and maintenance work. The project management literature can help them to divide the total development work systematically among the members and it may help to take up large projects.

1.5 *STRUCTURE OF THE THESIS*

This chapter has given a preliminary introduction about software industry and gives a overview of the thesis. The next chapter reviews the Literature. The Literature Survey focuses on different ideal team structures proposed by different researchers for better management control of software development process, the difference between types of software and some aspects of Indian software industry. The literature review also focuses on the ideal maintenance team composition for making maintenance work smooth.

The third chapter builds the research framework. This also gives idea on research methodology and method of data collection. and analysis. The fourth chapter gives analysis of the data collected. The data is analysed according to different division of work which is necessary to develop a software module. The fifth chapter gives a big picture of different types of team in comparison to ideal structure.

The last chapter gives conclusion of the study. It also gives limitations and shortcomings of this research work. Further ways to extend this thesis are also discussed in this chapter.

Chapter 2

2. LITERATURE SURVEY

The state-of-art software development tools cannot ensure better development practice. Powerful programming languages and fast compilers do not produce good software. Advanced development methods and software engineering practices may help, but offer no guarantees. In real the world of software and applications development, even the most rapid of rapid prototyping takes time, even mildly sophisticated systems need the contribution of multiple developers. Under these real circumstances, how the human resources of programming are organised and managed become crucial factors in the success or failure of development projects. Only good people, well organised and well managed to enhance their productivity and quality of their work, can produce good software."

(Constantine, 1993)

Software product is understood by only a few trained, technical minds. The mystic, the necessity, and the status surrounding computers in the eyes of management encompasses both the machine and people who program it. Programmers have capitalised upon management fear and inexperience with computers and have raised themselves to prima donna level in many organisations. The mystique surrounding programming-coupled with a poorly defined project organisation- makes project management a very formidable task."

(McClure, 1985)

2.1 INTRODUCTION

Broadly, the problem is of proper management methodology for software development. Thus the early literature scan was made with an intention to uncover material on managing software development, maintenance and Indian software industry. The literature was reviewed to find differences between Application software and System software. This chapter develops a background for understanding of software development team and build upto the research framework. Broadly the chapter is divided into four parts. The first part of this chapter is concerned with the nature of software and discuss why development methodology for software is different from other production processes. The second section of this chapter is focused on different formal team structures proposed by different researchers for better management of software development and maintenance. The third section describes the difference between System software and Application software. The last section describes Indian software industry. The next chapter of the thesis builds up research framework on the content of this literature survey.

2.2 THE NATURE OF SOFTWARE

This section attempts to highlight the nature of software that makes it difficult to apply general management techniques This section is an attempt at gaining some insight as to what is software.

The Software development is still considered an art. The use of tools is not prevalent in software development as in manufacturing. The management methodology followed for it is different comparing to other production processes because of it's unique characteristics (see Appendix 1). Because a lot of management myths exist for software development

which makes it difficult to supervise. This has been often called by the term " Software Crisis " (Pressman, 1989) (see Appendix 2).

The most celebrated books in the software arena is "The mythical Man-Month" by Brooks(1982). The book's 25th year since first publication (1969) was celebrated with a special edition and considerable coverage in the professional journal. His opening statement seems to be concerned with management aspects of software development. " In many ways, managing a large computer programming project is like managing any other large undertaking-in more ways than most professional manager expect." Further on, he states " This book is a belated answer to Tom Watson's probing questions as to why programming is hard to manage." Brooks answers the question in the first chapter (see Appendix 2). Among other points that characterise software programming, he lists the medium's tractability, a non repeating nature of job, the lack of control over the goal of work, or its circumstances and lastly, the rate of obsolescence. Brooks also emphasises proper management control of software production.

According to Pressman (1989) programming problems faced today are surprisingly similar to those faced in the sixties. Most of the software projects still fail. Schedules and cost estimates are usually overrun. Software seldom meets user expectation or requirement. The management approach could not keep pace with the change in technology. Because the same programming is done as ever. But in reality although the application domain may be same, but the demand for greater productivity, quality and criticality of software to strategic business objective has increased substantially.

The management thinks that well planned procedure like structured programming, waterfall model, Spiral model of development etc. exist for software development. It may well exist. But in reality it is never used. Because it does not reflect modern software development practice (McClure, 1985). In these circumstances the human resources management is the most complex task in the Project Management. Because the software development is still labour intensive.

Due to above stated peculiarities of software as a product, management control and monitoring is very much difficult in software development in comparison to other production process. So a formal structure of software development team is necessary. The requirements that should be fulfilled by a formal structure as defined by McClure in his book " Managing Software Development and Maintenance " are

1. Formalise the project so quality control standards , software engineering procedures, and management monitoring are possible.

2. Promote interaction between the user, the maintainer, and development team so that the software system will possess the qualities of usability and maintainability.

3. Solve the dichotomy between the programmer's preference to work alone and need to work as a part of a team to produce a well integrated system.

To achieve these above objectives many ideal team structures have been proposed. According to McClure (McClure, 1985) Egoless Programming team is the least formal

and oldest. Surgical team is the most specialised in its division of labour and is a refinement of the Chief Programmer team. In the next section of this chapter, the formal structures and division of labour in them is given. The advantages a structure provides and shortcomings associated with it are explained.

2.3 *FORMAL SOFTWARE TEAMS*

This section discusses about various types of software teams proposed by different researchers for development of software module. The main purpose of every structure is to make the process of software development visible to management and better quality control of developed software. Four different types of software development teams have been proposed, namely Egoless Programming Team, Chief Programmer Team, Surgical Team and Revised Chief Programmer Team. We will focus on different philosophy, characteristics, advantages and disadvantages associated with each type of structure. Our research objective is to find out similarity and deviation of actual team working from ideal structures. The various issues like

- Formal Leadership and management work of team including administrative work.
- Composition of team
- Designing, Coding, Testing of software module
- Way to establish Egoless Programming concept.
- Communication with other development teams and user of product
- Composition of Maintenance Team and communication with development team

etc. are discussed in different types of formal structure.

2.3.1 *Egoless Programming Team*

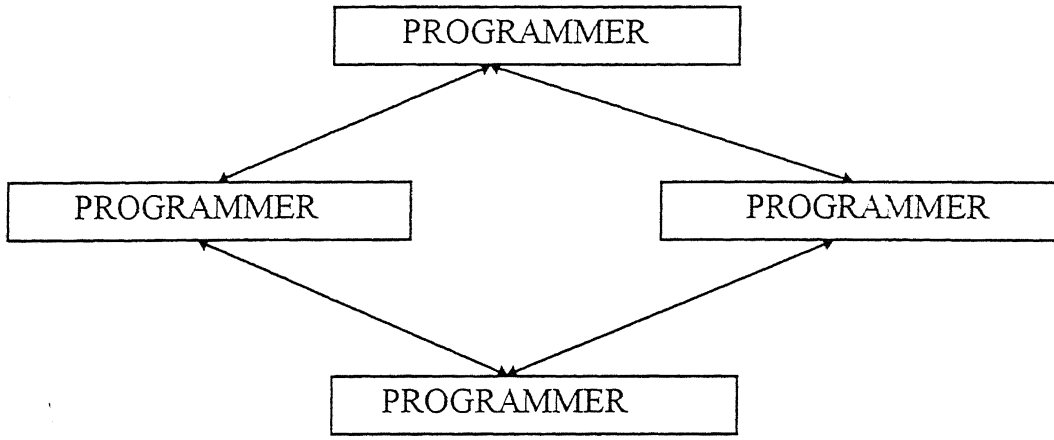


Fig 2.1 - Egoless Programming Team

It is the simplest of all ideal team structures having no formal division of labour. It is proposed by Weinberg (1971) and sought to open-up interaction and diminish conflicts over control, co-ordination, and expertise for people in the project team. Software people satisfied with working conditions and other team members would be easier to manage and develop more reliable system. The members work towards achievement of team goal without having any division of labour. It ensures an open, democratic work environment for its team members. According to the philosophy of Egoless Programming team, the extension of ego to programming practice is dangerous for the team organisation. The members should exchange their code to ensure egoless environment. The code exchange ensures error free and more likely readable code. The code exchange program makes the project visible to all the programmers and the critical decision can be made without depending upon single individual for critical decisions. It provides an excellent learning environment for team members, where they can learn new programming techniques and algorithms and discuss style and efficiency in a non-hostile atmosphere. Egoless programming team does not suggest specific functional responsibility for various team members, instead the responsibilities assigned to individual members are determined by the capabilities of members and team structure. The team leadership is rotated depending upon

the particular skills needed at various points in the project. All the functions like design, coding, integration work, testing is done by all the members together through mutual consensus between them. Although this structure improves system visibility, readability, and reliability, there are problems with its' democratic organisation. The drawbacks of this type of structure are

1 The responsibilities of each team member are not clearly defined. So the performance appraisal of members becomes troublesome making it difficult for management control and it may leads to lack of motivation among team members

2 In this type of democratic environment, no individual has the power to settle the disputes which makes it detrimental to system integrity. In some cases the decision may get postponed indefinitely due to endless debate among team members for all development stages like design, coding , testing etc.

3 Egoless Programming Team does not provide a leader to tackle the crises situation. Also the communication with user, management, other teams may become complicated since member's functions are not explicitly defined and may change during project.

The next type of team structure discussed is Chief Programmer Team.

2.3.2 *Chief Programmer Team*

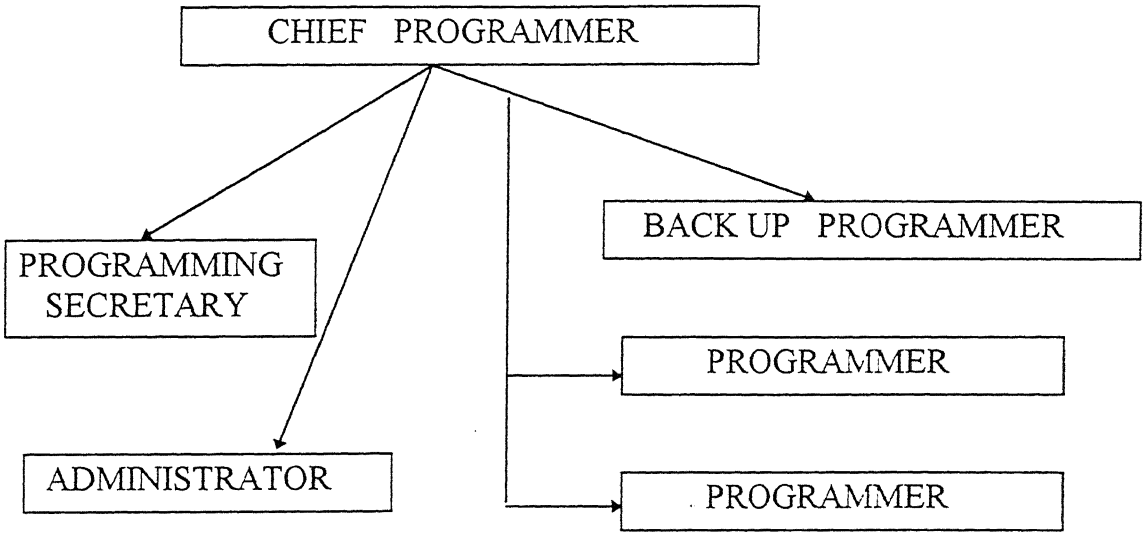


Fig 2.2 - Chief Programmer Team

The Chief Programmer Team (CPT) concept was proposed to adequately address the areas of management control, individual responsibility, motivation and evaluation, system integrity, leadership, decision making and communication. It was proposed by Baker (1972), was founded on a disciplined management approach where conflicts or uncertainties over what software work was to be done through formal lines of authority, control and expertise established within the team. In this type of paradigm, the Chief Programmer assumes the permanent leadership instead of rotating the leadership. So the control can be more easily implemented, communication with outside group is simplified and decision can be taken in deadlock situation. In the eyes of management , the Chief Programmer is directly responsible for technical success of project. His tasks include the development of system specification and design documents, the coding and testing of the critical portions of the system. In strictest form of CPT, he designs, codes, and tests every line of code in the system. If he can not do all the work, then he is expected to supervise the team members in their work. He should be a super programmer with at least 10 years of experience.

The Back-Up Programmer is the back up leader of the team. He is familiar with all aspects of the project and can take charge as leader of the team in case of any eventuality. His main task is development of test plan and research activities of Chief Programmer. He is also an expert programmer with several years of experience.

The Programming Secretary is the clerical member of the team whose principal responsibility is to keep all project documentation current and visible throughout the project. His task include maintenance of all program libraries, test data, test results and project documents. So instead of relying on code exchange as in Egoless Programming team, a program librarian is employed to make the most current version of project visible to team, to management, and to the user through out the project.

The goal of the CPT concept as proposed by Baker are

1 - Structure the software development work into specific tasks for assignment to technical specialists.

2 - Provide an environment in which state of art tools can be readily utilised.

3 - Keep the software product visible throughout its development process.

4 - Ensure that at least two team members understand every line of code.

5 - Provide a training environment for all the team members.

Although it is a developed structure in comparison to Egoless Programming Team, it has also come under certain criticism by Scacchi (1984) in IEEE Transactions on Software Engineering. These are as follows :

1. The CPT provides a challenging environment for Chief Programmer. But other team members do only menial work in comparison to him.

2. Since the Chief Programmer directly designs, codes, and tests the module , the quality of learning environment is not good for other team members. The structure imposed is very rigid. So other team members cannot develop their individual strength and remove weakness. It influences on the structure of software developed. Since the entire software module is product of a single mind, the module structure becomes an extension of thinking of the Chief Programmer.

3. The Chief Programmer enjoys too much power in CPT. He does all the creative work of software development and enjoys decision making ahthority in every aspect of team working. So there is very little motivation for other team members to excel in their profession. They always strive for super hero image of Chief Programmer. The quality of software developed may be low because it is a product of single mind. So Baker's proposal represents a troublesome replication of the principle of "Scientific Management" intended to deskill and downgrade the work of software professionals as well as encourage status-based segregation of project participants.

4. The main difficulties with CPT is non availability of such rare individuals like Chief Programmer who is expert in application area and programming. Also it is not possible for a single individual to execute all the functions of project management.

To get rid of these difficulties, the Surgical Team concept and Revised Chief Programmer Team concepts were proposed.

2.3.3 *Surgical Team*

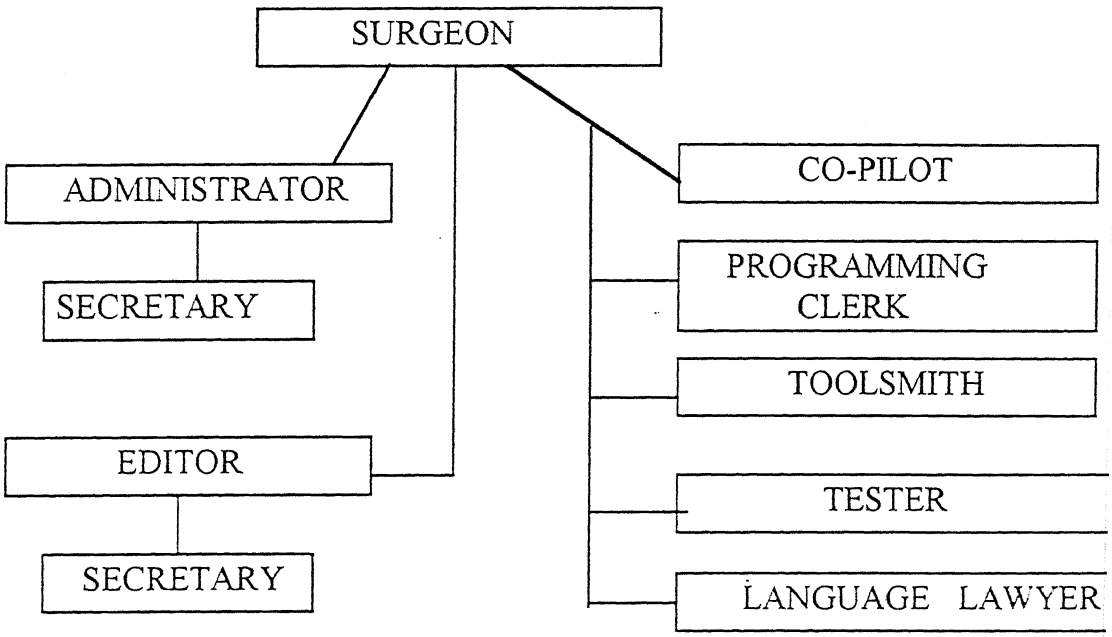


Fig 2.3 - Surgical Team

This team structure is proposed by Harlan Mills. It is the most complex team structure and division of labour is maximum among all team structures. According to Mills, the software team should be operated like a surgical team for executing an operation on human anatomy. In this paradigm, one does the development work and others help him to enhance his effectiveness and productivity. The team revolves around Module Leader who is called 'Surgeon'. Different persons of the team are,

2.3.3.1 THE SURGEON

He is also called Chief Programmer. He personally defines the functional and performance specification, designs the program, codes it, tests it, and writes its documentation. He has effective access to computing system which not only runs his test but also stores various versions of his programs, allows easy file updating and provides text editing for his documentation. He should be a talented man having at least ten years of experience, and considerable Application as well as System knowledge.

2.3.3.2 THE CO-PILOT

He is the alter ego of the surgeon, able to do any part of the job, but is less experienced. His main function is to help chief Programmer in design as a thinker, discussant, and evaluator. The Surgeon tries ideas on him, but is not bounded by his advice. He should interact with other teams of the same project, with maintenance team and with user if necessary. He knows all the code intimately. He obviously serves as insurance against disaster to the Surgeon. He may even write code , but is not responsible for any part of the code.

2.3.3.3 THE ADMINISTRATOR

The Administrator handles money, people, space and machines, and interfaces with the administrative machinery of the rest of the organisation. He should ensure that the Surgeon spends almost no time over these administrative matters. According to many software professionals, it is a full time and only job of a person if the project is very large, has substantial legal, contractual, reporting, or financial requirements because of user-producer relationship. Otherwise, one administrator can serve two teams.

2.3.3.4 THE EDITOR

For maximum clarity of documents prepared by Surgeon, the Editor must write it for both external and internal specification. He however, takes the draft or dictated manuscript produced by Surgeon and criticises it, reworks it, provides it with references and bibliography, nurses it through several versions, and oversees the mechanics of production.

2.3.3.5 TWO SECRETARIES

The Administrator and Editor both should be provided with secretary each. The Administrator's secretary will handle project correspondence and non-product files. The Editor's secretary help him in preparation of documents of software development.

2.3.3.6 THE PROGRAM CLERK

He is responsible for maintaining all the technical records of the team in a programming product library. He is responsible for both machine-readable and human-readable files. All the computer input goes to the clerk, who logs and keys it if required. He saves the programs in a chronological archive. He helps to transform the programming "from private art to public practice" by making all the computer runs visible to all team members and identifying all programs and data as team property, not private property. He is also expected to carry out integration work with the help of the Surgeon after taking the completed programs from the team members. He works as the interface manager in the team, so that the growing product is easy to manage.

2.3.3.7 THE TOOLSMITH

He ensures that all the software tools like file-editing, text editing and the interactive debugging services are readily available to team, so that a team will rarely need its own machine and machine-operating crew. The Surgeon is the sole judge of the adequacy of the service available to him. This is a full time job of a person in the team.

2.3.3.8 THE TESTER

The Surgeon will need a bank of suitable test cases for testing pieces of his work as he writes it, and then for testing the whole thing. The tester is therefore both an adversary who devises system test cases from the functional specs, and an assistant who devises test data for the day-by-day debugging. He would also plan testing sequences and set up the scaffolding required for component tests.

2.3.3.9 THE LANGUAGE LAWYER

He is a very knowledgeable person about the language used for development work of the software. He takes delight in mastery of the intricacies of a programming language. He is expected to help the Surgeon if some implementation difficulty arises. He suggests a neat and efficient way to use the language to do difficult, obscure, or tricky things. One language lawyer can service two or three Surgeons.

The Surgical Team solves the drawbacks of CPT by addressing various issues like

1 - The issues of team morale and individual recognition.

2 - The importance of a communication link and quality documentation.

Some criticisms of Surgical team are

1 Like CPT, the Chief Programmer has too many functions to perform and too much power.

2 The Surgeon's technical skills are preferred over the leadership skills. The Surgeon should be trained in project management skills as well as in technical skills.

The next structure proposed is Revised Chief Programmer Team (RCPT). According to McClure (1985) the RCPT is a better structure in comparison to Chief Programmer Team and Surgical Team structure.

2.3.4 *Revised Chief Programmer Team (RCPT)*

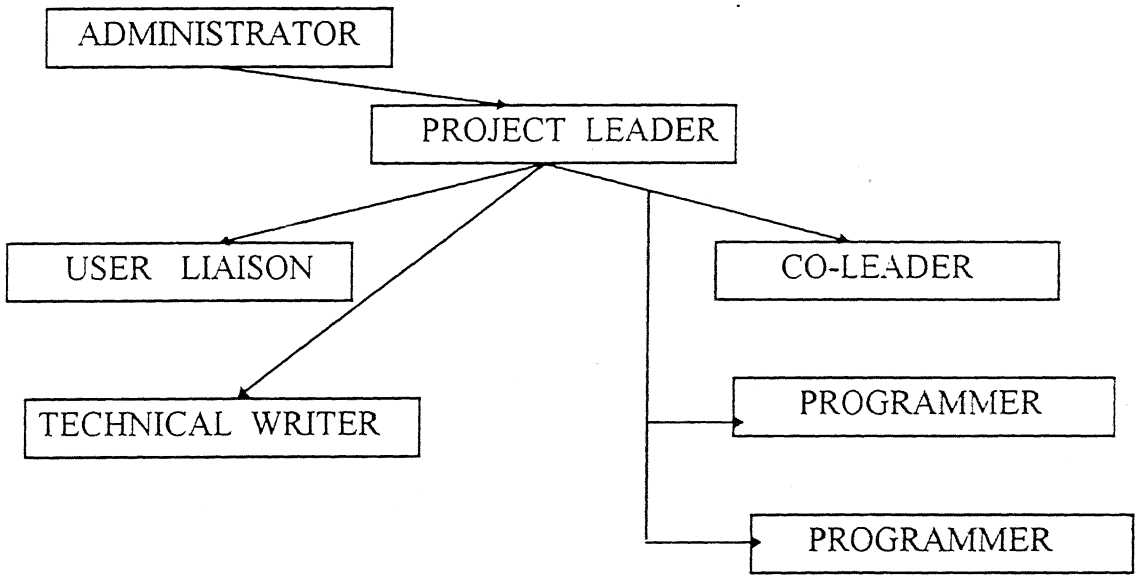


Fig 2.4 - Revised Chief Programmer Team

According to McClure (1985) the RCPT was proposed to reduce the responsibility and power of the Chief Programmer. In this structure, the Project Leader is below in hierarchy to that of Administrator who oversees the process of development of software module and

take care of the Administrative responsibilities. It is a combination of Chief Programmer Team and Surgical Team. The RCPT accomplishes the following

1 Enhance software product visibility so that the software can be subjected to periodic audit reviews during development.

2 Formalises the project organisation so that individual responsibilities and assignments are clearly defined

3 Establishes communication links between the project and the outside organisation including management, the user, others teams, and the operations and maintenance groups.

4 Encourage an open , egoless environment in which the state of art tools can be mastered and applied.

5 Avoids project dependency upon individual team members.

The different members of the team are

2.3.4.1 PROJECT LEADER

He is the technical leader of the team. He reports to Project administrator and all other team members report to him. The tasks of Project Leader are

1 - Determining project resource requirements with the project administrator.

2 - Developing and enforcing standards and procedures

3 - Producing functional specifications and design documents with the assistance of the co-leader and the user liaison.

4 - Assigning and managing all technical tasks.

5 - Reviewing all codes

6 - Reviewing the test plan and test results

7 - Writing internal system documentation

8 - keeping a project journal.

9 - Increasing the expertise of each team member

2.3.4.2 CO-LEADER

The Co-Leader is a experienced analyst rather than a "Super-Programmer". His tasks include

1 - Co-producing the system specifications and design documents with the project leader and user liaison.

2 - Developing a project test plan.

3 - Reviewing code and test results with the project leader.

4 - Investigating development problems

5 - Representing the team in technical meetings with outside groups.

6 - Co-ordinating project turnover with maintenance group.

2.3.4.3 PROJECT ADMINISTRATOR

The Project Administrator is the administrative head of the team. He performs the interface function with upper management. He is active in the project from inception through turnover. His tasks include

1 -Explaining to the team the project objectives and priorities as defined by management.

2 - Providing a suitable work environment and necessary resources for the team

3 - Handling the project budget.

4 - Handling the recruiting and performance evaluation of team personnel.

5 - Reporting project status and needs to upper management and the user.

2.3.4.4 USER LIAISON

He is the main communication link between user community and development team. He should have good application knowledge and communication skill. He should have also programming skill.

His main responsibilities are

1 - Responsibilities for the software product meeting user requirement and expectations.

2 - Interpretation of user requirements to technical members of team.

3 - Production of user documentation.

4 - Handling of public relations of the project team with the user community.

5 - Participation in testing, particularly in the construction of system and acceptance of test cases and test data.

6 - Co-ordinating project turnover with the team, the user, and the operations and maintenance groups.

2.3.4.5 PROGRAMMER

They are responsible for performing the coding and testing functions of the project. They function like egoless programming team among themselves and exchange code among themselves to impose egoless programming concept. Their functions are

1 - Responsibility for coding

2 - Adaptation of packages and existing routines for incorporation into the system.

3 - Responsibility for developing unit test plans and performing test.

2.3.5 *Maintenance*

For a typical heavy duty software, nearly 67 percent of total effort and resources are spend during the maintenance life cycle phase (McClure, 1985). So the process of maintenance should be more visible to management and more amenable to management control. The lack of formal organisation structure in many maintenance staff has contributed to problems of identifying individual responsibility, of providing adequate motivation, of preserving software quality, of making appropriate, timely support decisions, and of providing open communication channels within the maintenance group and with outside groups. An organisation should avoid software support dependency on one individual staff member.

So maintenance staff should accomplish the following

1. Enhance software visibility so that the software can be subjected to periodic quality control audit reviews.

2. Formalise the maintenance organisation so that individual responsibilities and assignments are clearly defined.

3. Establish communication channels between the maintenance staff and outside organisation including management, user groups, and the software development staff.

4. Encourage an open, egoless maintenance environment in which state-of -art tools can be mastered and applied.

2.3.5.1 MAINTENANCE TEAM STRUCTURE

The maintenance staff should be organised into teams of two or more members. A minimum team size of two is necessary to avoid vulnerable management situation of depending upon one particular individual for support of software system. Each team should be assigned software support responsibilities for a specified group of software programs. This identifies who is responsible for which software programs, allowing management to direct requests and questions to the appropriate team. Maintenance support responsibilities begin at the completion of the development phases when the software becomes a production system, and include all maintenance functions such as error detection and correction, adaptation to data and processing environment changes, and responses to user requests. However, support preparations begin during the software development or package selection phases itself.

According to McClure (1985) the Revised Chief Programmer Team is the most suitable one for maintenance function. The structure of the team should be same as Revised Chief Programmer Team at the development stage. To provide for the greater maintenance support needs of newly released software system, some members of the original software development should be recruited as members of the maintenance team. This may be considered as a temporary assignment until all software development work has been completed and system behaviour appear stabilised in terms of reliability, efficiency, and maintainability requirements.

The ideal structure of maintenance team structure is

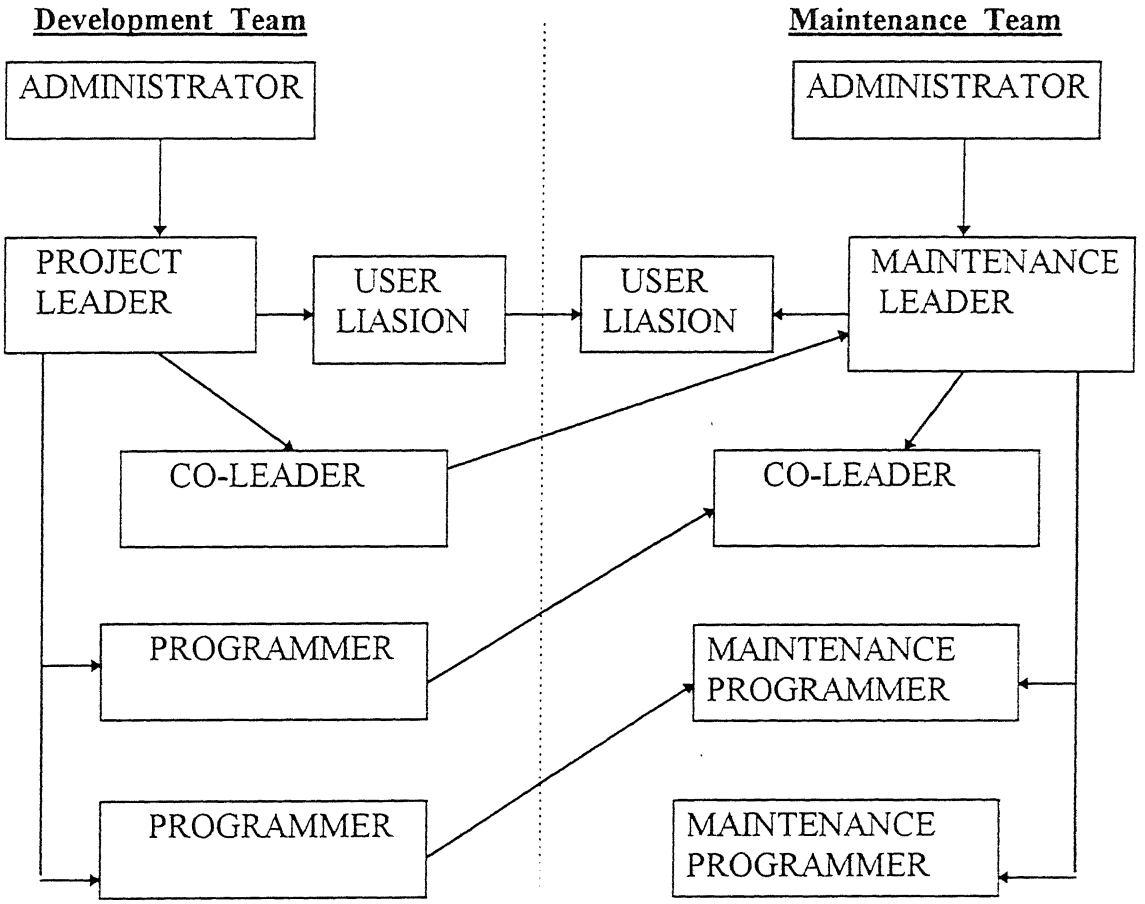


Fig 2.5 - Ideal Maintenance Team Structure

An ideal candidate for the Maintenance Leader is the Co-Leader of the original software development team. Having provided the communication link to the maintenance group during the development phases and being intimately familiar with the development phase of software, the Co-Leader is in an excellent position to direct maintenance activities, especially when the software is first released for use as a production system.

The User Liasion of the original software development should take over the same responsibility in the maintenance team. He is knowledgeable about the user need from the beginning of the development phase. So he can better understand the user needs and difficulties. Some of the capable programmers of the development team should be included

in the maintenance team. Because they have the first hand knowledge of the software quality and integrity of the software product. The rotation of development programmers into the maintenance area will help break down the barrier between the two programming factions. Because many consider the development group as an elitist group, and in many cases the trainees and programmers who are not able to keep up with technological advances are typically relegated to the maintenance group.

The next section of this chapter discuss the different types of software and difference between them.

2.4 *TYPES OF SOFTWARE*

Since our main research objective is to find appropriate team organisation for different types of software mainly between Application software and System software, so we are trying to uncover literature to find out the nature of various types of software. The software produced can be divided into two basic category of Application software and System software(Heeks, 1996). Application software is programs designed to carry out specific tasks, or Applications; as distinct from System software, which controls the operation of the total computer system. Typical Application software include word processing programs or database programs, whereas System software consists of compilers (which translate software language instructions into machine language instructions which the computer can obey) and operating systems (which act as the interface between the computer and computer user, and between the computer and the applications programs).

2.4.1 *Difference Between System Software & Application Software*

1 - System Software - It is a collection of programs written to service other programs. All the assemblers, compilers, and interpreters come under the category of system software that support the work of programmers. All the Fourth Generation Language Tools fall under the general rubric System software, the software used to invoke and run Application programs and programming aids, control the devices normally thought of as part of the computer system, often monitor the execution applications, and perform other useful tasks. System software makes the computer hardware invisible to the user. All the operating system components, drivers, telecommunication processors also comes under System software. All the softwares are characterised by heavy interaction with computer hardware, heavy usage by multiple users, concurrent operation that requires scheduling, resource sharing, and sophisticated process management. It is more complex to develop these type of software in comparison to Application software. The speed and performance carries a lot of importance for System software than Application software. All the characteristics make the System software more difficult to implement. The implementor enjoys more importance in these types of software in comparison to other softwares.

2 - Application Software - All types of Business Software, Real-Time Software, Engineering and Scientific software comes under the Application software. Business information processing is the largest single software application area. Discrete "systems" (e.g., accounts receivable/payable, inventory, etc.) have evolved into management information system (MIS) software that accesses one or more large data bases containing business information. Application in this area restructure existing data in order to facilitate business operations or management decision making. In addition to conventional data processing tasks, business software applications also encompass interactive computing.

Engineering and Scientific software has been characterised by " number crunching" algorithms. Applications range from Computer-aided design (CAD) , system simulation to biology to automated manufacturing. Although the Application software of these type are becoming complex, the level of interaction with hardware is very minimal in comparison to System software.

The next section discuss about the Indian software industry.

2.5 *INDIAN SOFTWARE INDUSTRY*

This section describes about the Indian software industry. The overall characteristics of the industry is described in the Introduction chapter of this thesis. We are discussing some issues related to our research framework. This section mainly discusses about nature of work taken by Indian software companies, Qualitative Employment Detail and state of technology used.

1- Onsite and Offshore work - Much of India's export work of developing custom software is actually carried out at the client's site overseas (onsite) rather than offshore in India(NASSCOM, 1996). According to survey carried out by NASSCOM, an average of 65 percent of export contracts were carried out wholly at the client site, while 35 percent contained some offshore elements. This translated into just under 75 percent of Indian software export development taking place overseas and only 25 percent in India. This was even true of work in the export processing zones, which were intended to be bases for offshore work.

2- Qualitative Employment Detail - According to NASSCOM's (1996) report, the average age within most software companies is mid to late 20s. Almost all staff had first

degree and many, particularly in software exports, had a postgraduate qualification. The table below shows the composition of workforce in Indian software industry (Heeks, 1996) in comparision to international standards.

Worker Type	Indian Software Industry (%)	World-wide software Industry (%)
Project Leader	3	12
Analyst/Designers	12	47
programmers	85	39

From the table it can be seen that the Indian industry is significantly 'programmer heavy' compared to the needs of certain export contracts. The actual situation is also worse than indicated by the table since many of the workers in any designed category have limited experience of that type of work. The Indian software industry is best suited to provide programming labour, for which there is no shortage of ' Raw Recruit ', while the lack of experienced project manager, analysts and designers has made it difficult for Indian companies to accept turnkey contracts (Heeks, 1996). According to Heeks, the division of labour is very slowly changing due to poor provision of higher skill training and experience, steady brain drain of skilled staff and because there was some tendency for it to be self-reinforcing. Most of the export work offered to Indian companies is for programmers. They therefore felt a reduced incentive to try to raise the skill profile of their workforce because they saw relatively little demand for analyst and the like. Staff with ten-fifteen years experience were still used only as programmer, unable to break through to more skilled occupations unless they left India. So it is becoming a shortage more of

Analyst and Analyst/Programmer than of programmers. It is a major determinant of structure of software development team in Indian software industry.

3- State of Technology - The use of Fourth Generation Language Tools are not prevalent in Indian software industry(NASSCOM, 1996). Very few companies have plans to adopt these tools in near future. The use of these tools will reduce the software development effort by ten fold (Pressman, 1989). There will be greater stress for more skilled work like System Analyst and less stress on coding and testing work with the use of fourth generation tools. So less labour will be required compared to the use of Third Generation languages. Yet the main skills bottleneck for the Indian software is not a shortage of programmers to write programs but a shortage of quality analyst able to decide what business was all about and how best to represent it in computing terms. So the increase of use of 4GLs is likely to aggravate the existing shortage of highly skilled software developer. So the wide spread use of these tools will cause India to loose it's competitive advantage in the long run.

This chapter created a background for understanding of the software industry and different formal team structures for better management control of software development and maintenance. The next chapter explains the utilisation of this body of literature to formulate the research objectives and selection of research methodology.

Chapter 3

3. PURSUIT AND PLAN OF STUDY

3.1 *INTRODUCTION*

In the first chapter, we began our search for appropriate team organisation for different types of software for better management control of development process. In the previous chapter on the literature survey, we developed an understanding of different types of software and different formal structures for smooth development of software. In this chapter we will define the research objectives and research methodology. The next two chapters analyse the data obtained.

3.2 *THE RESEARCH OBJECTIVES*

The broad objective of thesis is discussed in the chapter one. The main aim of the research is to find out the appropriate team organisation for different types of software team differentiated by number of people working in it and type of software broadly divided into two categories as Application software and System software. The modules are called as 'small' module when number of people working in it is equal to or less than four. Whenever the number of people working in it is more than four, it is termed as 'large' software module. Because from discussion with experts of software development, it was found that the division of labour in the team is visible only when number of people working in it is more than four. The literature provides ideal team structures for software development. But it does not illustrate difference of team structure for different types of software. So the main aim of this thesis is to compare actual team working with ideal

structure. The structure of the maintenance team and its' relationship with development team is also an aspect of this study for different types of software. So the main research objectives are

1. To find out the actual team structures of different software module teams and compare it with different ideal structures namely Egoless Programming team, Chief Programmer team, Surgical team and Revised Chief Programmer teams.
2. To find out different working style of software teams for different types of software like Application software and System software.
3. To find out the maintenance team structure and it's relation to the development team.

To fulfill the above objectives the questionnaire survey method was chosen as research methodology, the reason for which is explained in detail in the research methodology section of this chapter. The next section describes the research questions.

3.3 *RESEARCH QUESTIONS*

Since our objective is to find the actual working team structures and compare it with the ideal structures, the difference between various ideal team structures were taken into consideration. The ideal structures vary according to the responsibility and authority of the members in different aspects of software development. So the research questions were formulated according to different stages of software development(see Appendix 3).

The specific questions generated are related to :

- Formal Leadership and Management work of team including the administrative work..
- Composition of the team, qualification of Module Leader, Programmer. Technical Writer etc.
- Carrying out of the detailed design of the module structure.
- Solution of application difficulty in design including selection of algorithms etc.
- Participation of programmer in the team for different types of teams.
- Testing of software module.
- Integration of programs to complete the module structure.
- Communication with other teams of same project and with user of the software product.
- The structure and composition of maintenance team.
- Co-ordination and communication between maintenance team and development team.
- Function of Information Manager and Facilitator in the team.

The next section discuss about research methodology followed to address various questions discussed in this section.

3.4 *RESEARCH METHODOLOGY*

Since the objective was to find out working style of different types of teams and division of labour in them, the nature of this investigation calls for a broad based and survey oriented research methodology. Keeping this fact under consideration a questionnaire based survey of Indian software industry was carried out. The reasons for choosing this type of research methodology is

- Since we had to find difference between various teams, the case study method is too time consuming. Because a software development is lengthy process starting from requirement analysis to testing. Also very few percentage of software development process satisfy the pre-determined time schedule.
- The software development process is a highly specialised work in which only the team members are capable of making out the nature and complexity of the problem. Even it is difficult for management to know about criticality and complexity of development process. So the case study method was difficult to conduct. So questionnaire based research was deemed suitable for it.

To overcome the disadvantages of questionnaire method like no opportunity for probe, no control over who fills out the questionnaire, low response rate (Nchmias & Nachmias, 1985) it was decided by researcher to distribute and collect the responses personally. So it was possible to clarify any doubts about the questionnaire to the respondents. Also it was possible to collect response in a very small span of time saving the precious time.

3.4.1 *Sample Of Company*

The sample of company selected is situated in Noida and New Delhi. The main criteria for their selection was geographical location of New Delhi & Noida. These companies are very closely located in various Software Technology Parks, Export Processing Zones and Industrial areas. The sample size chosen was 30. The total no of companies responded is 25. Considering the nature of our study, it was a satisfactory size of sample.

3.4.2 *Respondents*

According to Kraft (Kraft, 1979), there are basically two different kinds of software workers who are called "System Analyst" and "Programmer". Analysts are generally responsible for designing whole, complex data processing system rather than parts of larger one as programmers do. As a result, they tend to be "customer's men" go between as much as technical specialist. Their position is ambiguous because their work contains elements of the manager and salesman as well as technical export. All the Module Leader, Project Leader and technical experts comes under the heading of System Analyst. The Analyst's role as " conceptual architect " contains elements of both the technical specialist, the super programmer, and the managerial "generalist", the intermediary who works out the specifications set by the customer and which are to be met by the programmers who do the work. So he can be either a high level technical expert or a go between customer and developer depending upon the work.

The second kind of software professionals are called programmer. They do the slow work of writing down the code which expresses the program. It is the least creative and low value added work of software module development. The problem given to them may be

specific or general depending upon the type of project and capability of individual programmer. All the Technical Writer and Tester of module comes under the heading of programmer in division of labour in the software industry. They are expected to contribute in designing the structure at module level without any participation at the project management level. They stand at first step in the ladder of hierarchy in software industry.

So it was decided that one person from each division of labour from each organisation will be ideal for collecting response. It gives the opinion of two sections of the workers in the software industry. So necessary variations of opinion can be taken into account in the data collected. Since the software organisations are very small in size, there is no difference between System Analyst and management of the organisation. So the data collected from companies takes into account of every one involved in the software development process.

In the next section of this chapter discusses about the design of the questionnaire.

3.5 *QUESTIONNAIRE DESIGN*

In order to achieve the research objectives a data collection tool was designed, which helps in collecting the requisite data. Different questions of the questionnaire addresses different aspects of the functions required for development of software module (see Appendix 3). Because the ideal team structures vary in responsibility and authority of members in different functions of software development process. Since our objective was to find out actual working team structure for System software and Application software, every question seeks answer for these two types of software. After consulting the expert in the field of software development, it was found out that the team structure is also

dependent upon the no of people working in it. It was found from discussion with them that the division of labour in the team is practically visible only when the number of people working in the team is more than four. So the teams were divided according to number of people working in it. A team is called 'small' team when number of people working in it is less than or equal to four. The team is called 'large' team when the number of people working in it is more than four. So after dividing the team according to type of software and size of team, we finally arrived at four different types of team. The four different types of team are 'Small Application Software Module Team', 'Large Application software Module Team', 'Small System Software Module Team'. All the questions in the questionnaire addresses these four types of software module team.

The first aspect of the questionnaire is concerned with formal leadership and management of the module team. It was tried to find out necessary qualification of Module Leader for different types of team differentiated according to type of software and size of team.

The other aspect of the development covered is designing of the structure of module. Design translates the requirement for the software into a set of representations (some graphical, others tabular or language based) that describe data structure, artitechture and algorithmic procedures. It is the high value added work of software development. It was tried to find who designs the detailed structure of module and who is expected to solve application difficulties of application area in the design phase. The Participation Index of programmers was tried to find out for different types of software teams.

Documentation of the development work is required for clarity of the development process and making manual to be used by the user of the software. The internal documentation is necessary for clarity where the internal structure of software module and positioning of different algorithms in the structure is described in detail. It is a detailed document of software including both internal and external functions. During maintenance phase the internal documents helps in detecting bug at particular position of the software. The external document describes the external functions of the software. The external document describes in detail about the use of the software. It accompanies the software to the user site. Different ways in which this function is carried out in team is tried to find out through this questionnaire. The qualification of Technical Writer (if he exist in the team) necessary for making technical documentation for different types of software was tried to get form the research work.

Design representation must be translated into an artificial language (conventional programming language or a non procedural language used in the context of the 4GT paradigm) that results in instructions executable by the computer. The coding step performs this translation. It is the low value added work in the development process and is the least creative. Usually very junior people in industry do these kind of work. The effort was to find the qualification of programmers, the other work they do besides coding work. How the correctness and reliability of their code is ensured? How much the senior people in the team are involved in coding work? were also tried to find out.

Since the software product is ultimately used by the final user, it is very important to know the requirements of the user. So frequent communication with the user is necessary to

ensure the quality of software. The external specification is determined according to the user need. The communication with other teams of same project is necessary to ensure conceptual integrity of the software. The questions were designed to find out who is involved in these functions. Also the co-ordination and communication between members of the team is necessary to convey the design ideas from artitechture to implementor. Different mode of communication prevalent in the team for various types of software was tried to know from the responses given by respondents in various organisations.

Interface Management concerned with structure of module and relative position of programs in it. The programs are integrated after removing the bug to complete the structure of the module. So one person in the team should be aware of structure of module and relative positioning of programs in it. It makes the integration work smooth and the members remain clear about their work. How this work is addressed in the team? for different types of team was tried to know from the collected responses.

Testing of the module is done to check the conformance of module to the specifications set during the design phase. It helps to uncover defects in function, logic and implementation. For many complex software it involves checking speed and performance of programs with the specifications. Only surgical teams boast of a specialised tester function in the team. No other team assign a member for specialised testing work. So some questions in the questionnaire included to focus this aspect of software development.

Maintenance phase starts prior to release of the software and continues throughout its useful life. During maintenance errors are corrected, adaptations are effected, and

enhancements are implemented. For a typical software nearly 50-70 of budget and effort is spared in the maintenance phase. The maintenance work is very much dependent upon the process and methodology followed in the development phase. So a close co-ordination between development team and maintenance team is required for making the maintenance process smooth. The maintenance people should get a first hand knowledge of the development process. So either same development team should take maintenance work or a person in the development team should act as a bridge between development team and maintenance team. The mode of communication bridge between development team and maintenance team was tried to find out for different types of software module.

3.6 *VALIDITY & RELIABILITY TESTING OF QUESTIONNAIRE*

The Validity and Reliability of the questionnaire was tested after making the questionnaire. The Validity was tested mainly by the method of **face validity**. The questionnaire was given to four persons of different companies to give their judgment. The questionnaire was modified slightly according to their valid suggestions. Finally it was given final verification by former C.E.O of Cadence Systems Design.

The Reliability of questionnaire was tested by subjective judgment of responses rather than following any numerical method. Because the options attached with the questions were different facts rather than scaling a single item. After collecting response from four responses from two organisations, the two responses of every company were compared with each other to check the reliability.

3.7 *DATA ACQUISITION PROCESS*

The companies visited are situated in Noida, New Delhi and Gurgaon. The data was collected by personal visit of researcher to every company. The researcher had to meet the Chief Executive Officer or Human Resource Development authority in different companies. The plan and purpose of the study was explained to them assuring full confidentiality of data about their company. It. was requested to concerned authority of company to give the questionnaire to one System Analyst and one Programmer of the company to give their response. It was requested to ensured that the respondent System Analyst should have minimum five year of experience in his position and the Programmer at least one or more year of experience in the company. The questionnaires were distributed in the companies and the respondents were requested to fill it in their free time. All these questionnaire were collected after one or two days of distribution to respondents.

All these companies are situated in various Software Technology Parks. Export Processing Zones and Industrial areas. The close proximity between various software companies were of great help in collecting the responses. The study has been a confidential one and thus name of the companies have not disclosed in this piece of writing. So the nature of team structure and variation of them from ideal structure have discussed in finding of analysis of data without giving the names of the companies. The companies were quite co-operative in the research work.

In total thirty companies were visited by the researcher out of which twenty five companies responded. The total no of responses collected are forty six. Twenty four

respondents are of System Analyst rank with having no of years of experience more than five and twenty two respondents are of Programmer rank with no of years of experience more than one year.

3.8 *STATISTICAL ANALYSIS OF THE DATA*

Since our aim is to see the difference between the Small software module team, Large software module team and between the System software module team, Application software module team, the statistical analysis have been applied to test the collected data wherever possible. Mainly Non-Parametric method of statistical testing is applied to the data. Since there is no information about the distribution of the population. Under the Non-Parametric testing techniques between the two independent samples, the following tests are available (Siegel, 1956) : -

1. Binomial test.
2. Fisher exact probability test.
3. Chi Square test.
4. The Median test.
5. The Mann-Whitney U test.
6. The Kolmogorov - Smirnov Two Sample test.
7. The Wald - Wolfowitz Runs test.
8. The Moses test of extreme functions.
9. The Randomization test for two independent samples.

Since in every case we have only frequency of different options, the Chi-Square test was deemed suitable for this type of statistical analysis. The conditions required for application Chi-Square test are (Siegel, 1956)

1. In every case, the total sample size should be more than 40.
2. The expected frequency for more than 20% of the options should be more than five.

Since both conditions are satisfied by the collected data, Chi-Square test was applied for the statistical analysis. The significance value ($\propto$) chosen is 0.1 for every test. It is a general practice to choose value of $\propto$ as 0.1 or 0.05.

Only for testing the significance of Participation Index for different types of software team, Z-test has been applied. Because the information in the data is sufficient enough for carrying out Z-test.

The next chapter analyse the collected data.

4. Analysis of the Data

4.1 *COMPANIES VISITED*

The companies visited are situated in Noida, New Delhi and Gurgaon. They are situated in different Software Technology Parks, Export Processing Zones and Industrial areas. Out of 25 companies visited, fourteen are situated at Software Technology Parks, seven are at Export Processing zones in Noida. Others are situated at different industrial areas in New Delhi. Seven companies were developing software in Application area and five companies were developing System software only. Thirteen companies are developing both System and Application software. From each company, two responses were collected. Since there are two main division of labour among computer professionals, namely Programmer and System Analyst (Kraft, 1971), one response each from one group was collected. The response was collected from one System Analyst of company who was in the rank of Project Leader. The Programmer of company with minimum of one year of experience was considered suitable for taking the response. Forty-six responses were collected from these 25 companies. Out of them, twenty four were filled by System Analyst and twenty two were filled by Programmers having more than one year of experience.

Sixteen of the respondent companies have employee strength of less than 40. Only six of these companies have employee strength greater than 100. When asked about the turnover of the company, 20 out of forty six respondents were not sure about the turnover of the company in which thay are working. Because 11 out of 25 companies were owned by

different oversea companies. So they could not provided exact data about turnover of the company. Out of twenty five companies, twenty companies are very young with average organisation age of less than three years. Since the market for software products is very limited in India, except for four of the companies, all others are developing a part of large project for some other developer abroad. In many small organisations the professionals were developing software at the client's site. All these 46 responses have been analysed to derive conclusions about structure of different types of software teams.

4.2 *SIZE OF SOFTWARE DEVELOPMENT TEAM*

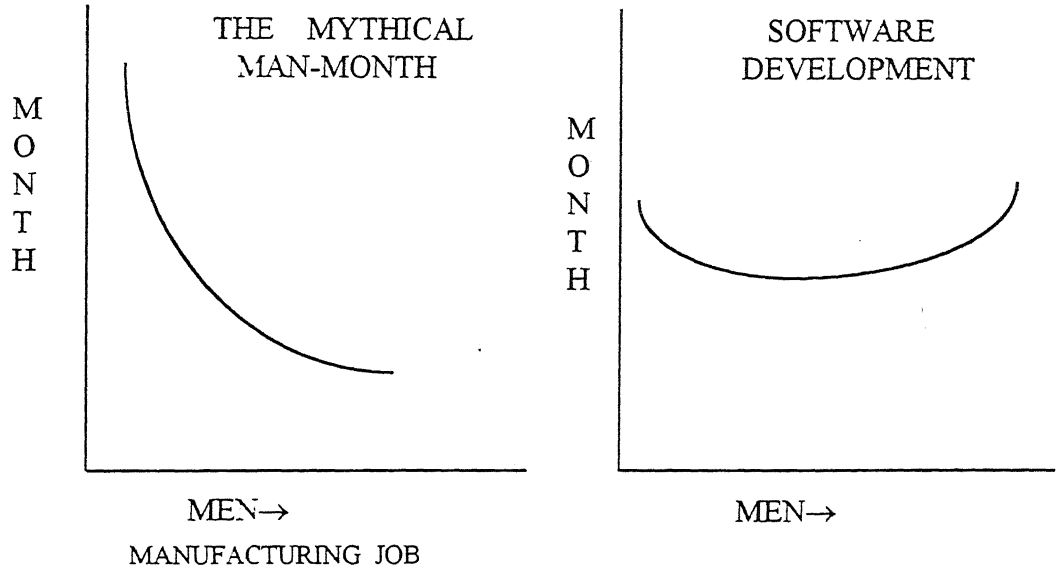
According to 41 respondents of the software development projects, the maximum size of software teams is less than 10. From 46 responses collected, only five of them felt that the number of persons working in the software project can exceed 10.

4.2.1 *Discussion*

According to the respondents where the number of people exceed 10, majority of team members were not expected to contribute towards the success of the project. These people were in their training period and expected to learn from the other members working in the project and contribution to the overall project was secondary task for them. According to the respondents, the cases where the number of persons working in the modules was greater than five is rare in comparison to small modules. According to respondents from large organisations (Where the employee strength of organisation exceeds 250), when the module is very large, complex and critical the number of people working for the development work exceeds 5. Most of the respondents felt that the development in large size modules is very rare in comparison to small size modules. It is due to the fact that the

16 out of 25 respondent companies has employee strength of less than 40. So they are unable to develop very large software projects. Only respondents from some big organisations felt that work should be managed in large modules. As it was expected, the size of module team is less than 10 in 41 out of 46 cases. Because as the size of the team increases, the effort in co-ordination and communication between members of the team increases exponentially (Brooks, 1982). A member of the software team can not work independently without communicating with other members of the team. The typical man-month analogy can not be applied to the software development effort. Because unlike manufacturing job the effort and time required for the job increases with increase of manpower after some critical point (Brooks, 1982) due to requirement of additional communication effort between the team members in addition to development work.. The diagram on next page shows the difference between manufacturing job and software development in relation to addition of man power. From the diagram it is clear that the time required for completion of the software project increase with addition of man power after some critical point due to added communication effort.

Time verses Number of Workers



Source : The Mythical Man Month by F.Brooks

The software development process is still considered as an art. The size of the team is not dependent on the type of software. So the optimum size is same independent of Application and System software. In three companies it was found that only Application software is managed in large modules and System software modules are managed in small teams. This is due to fact that a very few companies design and develop the large System software. They are only developing small part of a big project for another developer abroad. So the range of expertise required is less than what is required for a big and complex projects. So we can conclude that software development process is better manageable in small teams rather than in large module teams.

4.3 *FORMAL LEADERSHIP AND TEAM MANAGEMENT*

According to Surgical Team paradigm and Chief Programmer team paradigm, every software team should have a formal leader who is called 'Module leader'. His functions

vary according to different paradigms. Only Egoless Team concept denies presence of a formal leader. Question no 5 & 6 are concerned with this aspect of software development team. For large modules, the percentage having a module leader is 82 % where as in case of small modules the percentage is less than 50%. The type of software Application or System, has no influence on this aspect of the development team (Refer Table 4.2 of Appendix1). Figure 4.3.1 & 4.3.2 shows the frequency plot of number of cases having a formal Module Leader for different types of software.

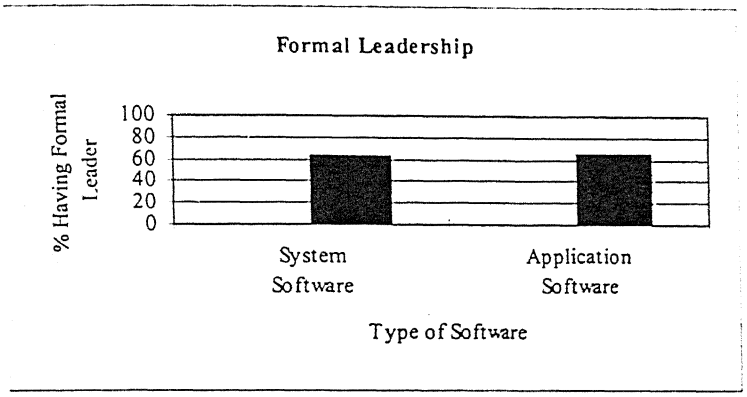


Fig 4.3.1

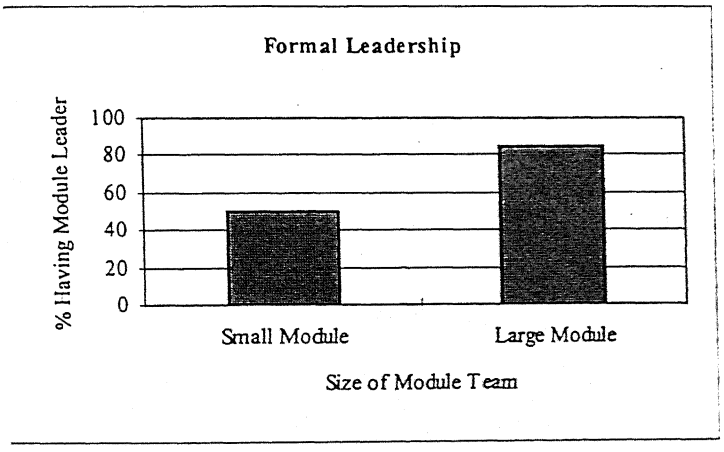


Fig 4.3.2

4.3.1 *Discussion*

From the data it was found that the formal leadership is more prominent in case of large software modules in comparison to small software teams. The Fig 4.3.1 & 4.3.2 shows the percentage of cases the formal leadership exist in the team for different types of software teams. According to respondents, it is a general practice to assign Project Leader in charge of small teams and without having a Module Leader. But from the data (Refer table 4.2 of Appendix 1 of the chapter) it could be seen that there is presence of Module Leader for 45% of cases for small Application software modules and 58.2% for small System software modules. When inquired about this aspect, the respondents felt that many small projects consists of only one module and there is no difference between Module Leader and Project Leader. In some cases of large modules there is absence of Module Leader and Project Leader takes charge of more than one team. This was the case for very small software organizations where the total technical employee strength is less than 20. They were unable to keep more qualified professional in their organization.

4.3.1.1 QUALIFICATION OF MODULE LEADER

The qualification of Module Leader varies according to the type of software. Nearly 85% of Module Leaders of System software were having a B.Tech, B.E, M.Tech degree in Computer Science and Electronic Engineering (Refer Table 4.3 of appendix 1 of the chapter). This is as making of System software is more complex in comparison to Application software. It involves heavy interaction with machine and the developer is expected to have a good knowledge of hardware. The Module Leaders of Application software were having a B.E , B.Tech , M.Tech, MCA or Msc degree. In 20% of cases the

Module leaders of Application software were having degree in Computer Science or Electronic Engineering. According to the respondents, the qualification of person carries secondary importance in comparison to capability for taking these assignments.

4.3.1.2 EXPERIENCE OF MODULE LEADER

The Module Leaders of System software are having more years of experience than Application software (Refer Table 4.4 of Appendix 1 of the chapter) and the respondents felt that they should be more capable than their counter part in Application software. The experience these people were having varies from 1yr to 7yrs. But according to the respondents, the capability of person carries prime importance for taking these assignments and experience is secondary in comparison. According to the respondents of the organisations which are developing both System software and Application software, more capable persons having more number of years of experience take charge of team as Module Leader of System software. In many cases it was found that people having only one year experience take the charge of module as its leader. It is due to the fact that the Indian Software industry is in very nascent stage and has yet to attain maturity (Heeks, 1996). The average employee age in Indian Software industry is only 29 (NASSCOM, 1996). The majority of its employees are programmer and only 3% of its' employees are System Analyst who are capable of taking charge of Module Leader (Heeks, 1996).

4.3.2 *Administrative Decisions*

Question number 9 of the questionnaire addresses this aspect of the software development. The administrative functions include allocation of resources, getting money from organisation, addition of manpower in the middle of development work etc. It was found

that project leader and top management take the administrative decision in 84% of the cases in small modules, where as in large modules the project leader in consultation with top management or project leader in consultation with module leader takes the decision in 78% of the cases. Fig 4.3.3 & 4.3.4 shows the pie plot of the ways of decision making in small and large modules. The type of software of System software and Application software has no influence on administrative decision of the module team (Refer Table 4.5 of appendix of the chapter).

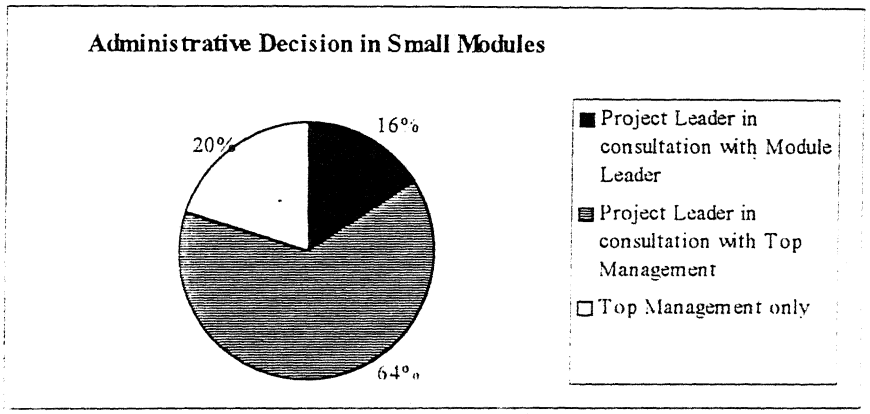


Fig 4.3.3

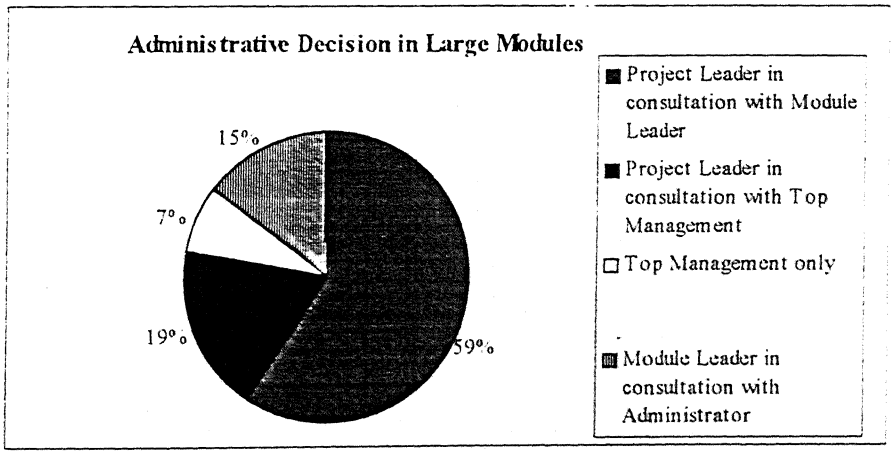


Fig 4.3.4

4.3.2.1 DISCUSSION

These Module Leaders of software teams are not expected to take administrative responsibility like allocation of resources, getting money from organization during time of need, addition of manpower in the middle of development work etc. of the team. The responsibility is mainly taken by Project Leader and Top Management in small modules and Project Leader and Top Management in consultation with Module Leader in large modules. These matters do not comes to team level for decision in 16 of 25 respondent organizations where total number of employees is less than 40.

It was observed that the specialist administrator function does not exist in software module teams as suggested in Chief Programmer Team paradigm and Surgical Team paradigm. Only in three companies there was presence of specialized administrator in large module teams. It is due to fact that these three companies are very large organisations having their branches in many countries of the world. So the administrative work is substantial and a specialized . It was found that one administrator takes charge of two or more teams. They were helping Project Administrator for settling administrative problems. But in no case the Administrator heads the team as proposed in the Revised Chief Programmer Team paradigm. In all these three organizations the human resources division people were taking the charge as administrator of the team.

4.4 *DESIGN OF MODULE STRUCTURE*

Design of software is the most important aspect of development of software module. Question number 7&10 of the questionnaire address the design aspect of software development. Question number 11 discusses about the Participation Index* of programmers in the module team. The algorithms to be used , the interface between the programs is decided in the design phase. The external specification and internal specification of the module is decided while designing the module. It was attempted to find out who is involved in designing of software module. The focus was only on module level without considering the total software project as a whole. Also it was attempted to find out the involvement of team members in the design process. The figures 4.4.1 & 4.4.2 show the ways in which the design work is carried out for different types of software modules.

Designing of Application Software Module

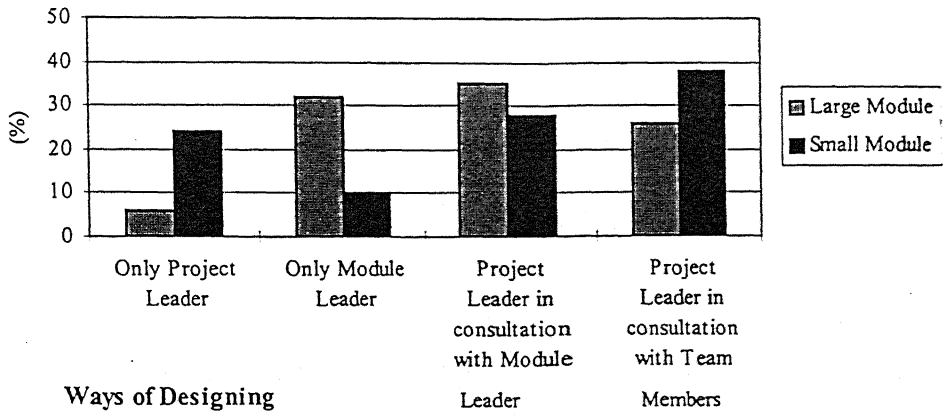


Fig 4.4.1

* The Participation Index of the programmer in the team is defined as the degree to which a programmer perceives his/her involvement in designing process of the software. The Index varies from 0 to 6 for different options stated in the question number 11 of the questionnaire. See Appendix 4 for the questionnaire.

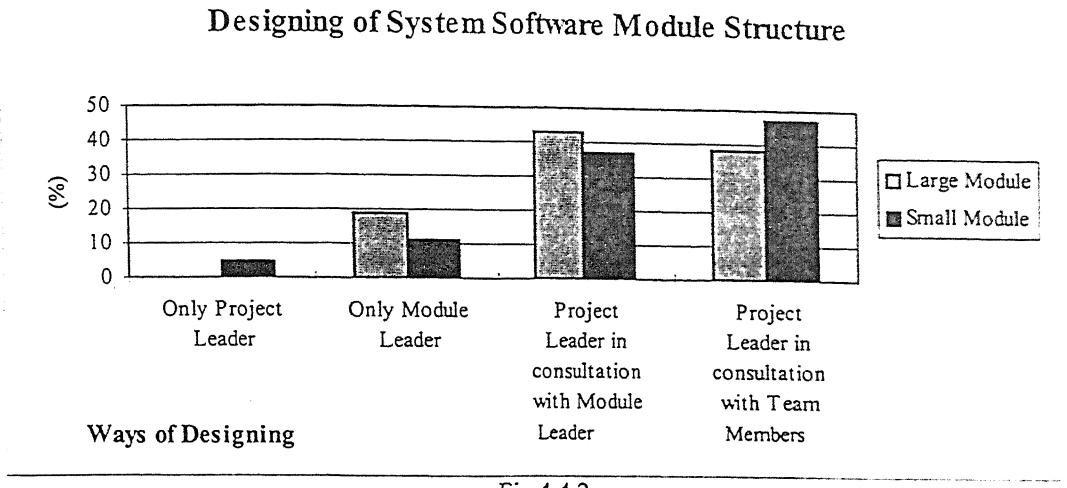


Fig 4.4.2

In general, the programmers of System software are better qualified and capable than Application software. Question number 18 of the questionnaire asks about the educational qualification and number of years of experience required for the programmers of the software module. It was found out that the participation index* of better qualified programmers are higher than others. In Application softwares where the programmers are better qualified, their Participation Index was found higher than others. In four cases of System software nobody was involved in design process except Module Leader. The respondents felt that if the Module Leader is a very capable analyst, then he is given the responsibility of designing the module himself. Difference in Participation Index of programmers was observed between small module teams and large module teams. In case of small team the participation index is more than Large team. The difference between designer and programmer is more distinct in Large software teams. The table shows the participation index of programmers for different types of software teams. Question number 11 of questionnaire attempts to find Participation Index of programmers in different types of software module.

Participation Index

TYPE OF SOFTWARE MODULE TEAM	MEAN (PARTICIPATION INDEX)	STANDARD DEVIATION (PARTICIPATION INDEX)
Small Application	3.8	1.8
Large Application	3.7	1.9
Small System	4.56	2.0
Large System	4.36	1.91

Table 4.4.1

4.4.1 DISCUSSION

From the diagrams it is evident that the programmers are involved in the detailed design of the module structure. According to the respondents the programmers participate in design according to their qualification and experience. According to Chief Programmer team and Surgical team concept, the design of module structure should be product of single mind to achieve conceptual integrity. Only Egoless programming team talks about the participation of programmer in design. But in actual practice, the members give their view in design process. It helps to remove certain shortcomings of Chief Programmer team and Surgical team like the Module Leader takes all creative work of development process and there is very little scope for other members to learn in the team environment. The Participation Index of programmers of Small module team is higher in comparison to Large module team. So distinction between designer and implementor is low in Small software module in comparison to Large module. Also the Participation Index of programmers of System software is higher in comparison to Application software.(Refer Table 4.12 of Appendix 1 of the chapter)

The knowledge of application area is the prime requirement for designing of software module structure. The blend of appropriate software engineering practice and technical knowledge leads to the development of the software product. So the development team should always have source of technical expertise. Question 12 tries to find out how the technical expertise is availed by the team. Fig 4.5.1 to 4.5.4 show different ways to address the technical difficulties faced by the team for different modules.

NOTATIONS USED FOR FIG 4.5.1 TO 4.5.4

a1 = P.L is expected to know a lot about application

a2 = M.L is expected to know lot about application

a3 = A team member is specialist in application

a4 = The help of specialist is taken during requirement

a5 = All the above options

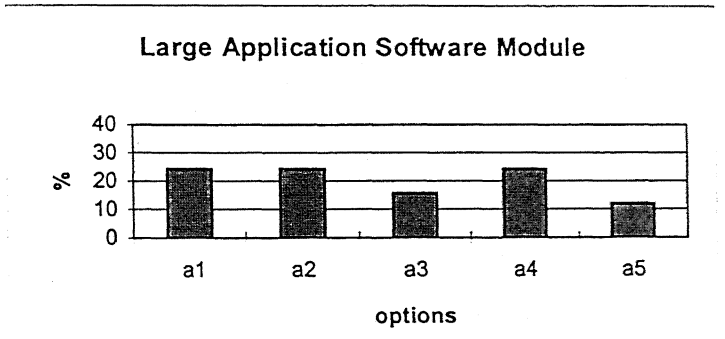


Fig 4.5.1

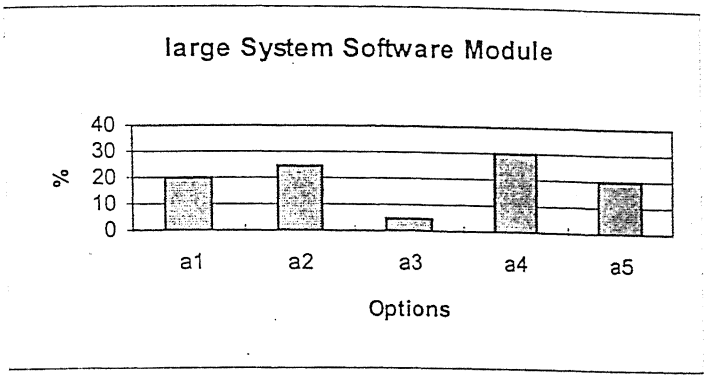


Fig 4.5.2

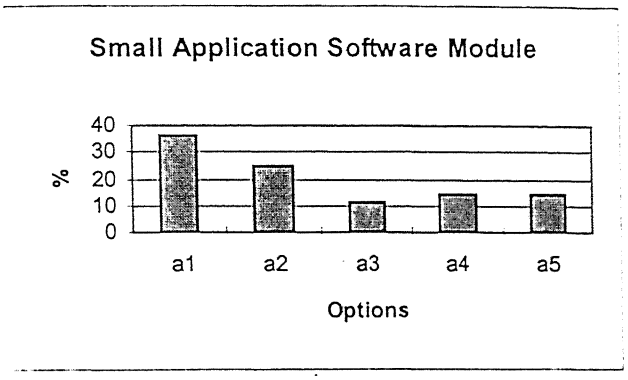


Fig 4.5.3

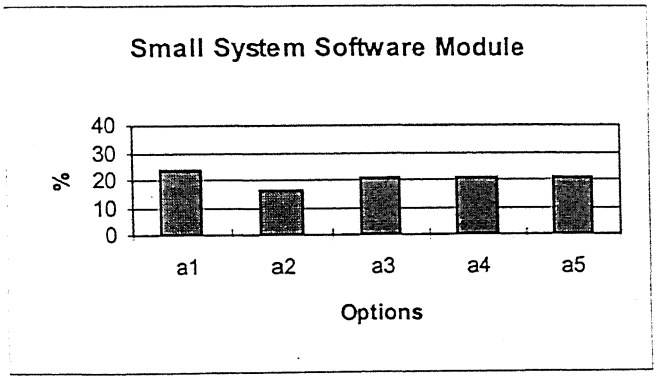


Fig 4.5.4

4.5.1 *Discussion*

From the bar charts it can be seen that in case of Application software Project Leader and Module Leader if he exists in the team is expected to be a expert or specialist in the application area concerned. From the statistical testing applying Chi-Square test it was

found that the presence of specialists is more possible occurrence in case of large Application software module than small Application software module (Refer Table 4.8 of appendix1 of the chapter). But according to all the respondents the presence of specialists in team is dependent on the employee composition of organisation, capabilities of Module Leader and Project Leader and complexity of the project rather than on particular type of project differentiated as Application software and System software and size of software team. So it can be concluded that the presence of specialist is not dependent upon the type of software.

4.6 *TECHNICAL WRITING*

It is the creation of technical manual of the software module for clarity. Question number 13 & 14 focuses on the technical writing aspect. The manual describes all the external specification, internal specification, algorithm used for making of software etc. The Project Leader or Module Leader is expected to produce original manuscript and technical writer is expected to rework it, provide it with references and bibliography, nurses it through several versions, and oversees the mechanics of production. The Technical Writer was found in every software module. But he was not expected to review and criticize as proposed in the Surgical team concept. Only in small modules the editor or technical writer is expected to do some coding work (Refer table 4.9 of appendix of the chapter). For large modules it is a full time work of a person. Editor of the team is a technical person of the application area concerned. There is a lot of importance for documentation work because of high employee turnover rate in industry. Every team member is required to make document of his coding work.

4.7 *REVIEW OF DESIGN*

Question number 13 of the questionnaire focuses on this aspect of software development. The respondents could not give appropriate response for this question. All of them felt that the review function is dependent on the capabilities of Module Leader, Project Leader, Programmers, criticality and complexity of software and sophistication level of user of product. If the user is sophisticated and capable, he may also participate in design review of software. So the design review function does not depend on type of software module differentiated as Small System software module, Small Application software module, Large Application software module and Large System software module.

4.8 *TESTING OF SOFTWARE MODULE*

Testing function is necessary to ensure that the software module conforms to the external specifications set up in the requirement analysis stage of software development and internal specifications in the design stage. For more complex programs the speed and performance is also tested during this stage. Question no 16 of questionnaire attempts to find out answer to this question. No significant difference was found between two types of software. Some difference was observed between small and large modules. The diagrams shows the difference between them. In small modules, for 50 % of cases any team member does the testing work and in 40 % cases members of other team are called to test the module. But in large modules only in 23 % of cases any team member test the module and in 65% cases the members of other team test the module.

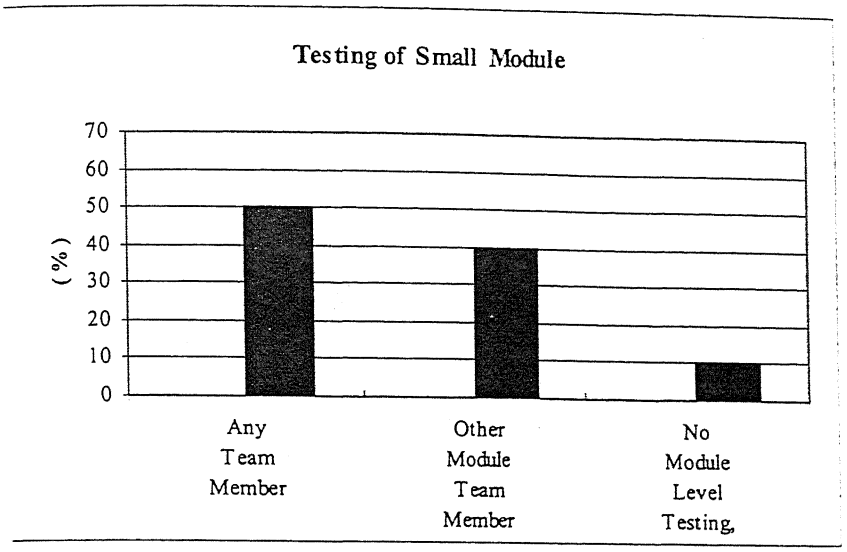


Fig 4.8.1

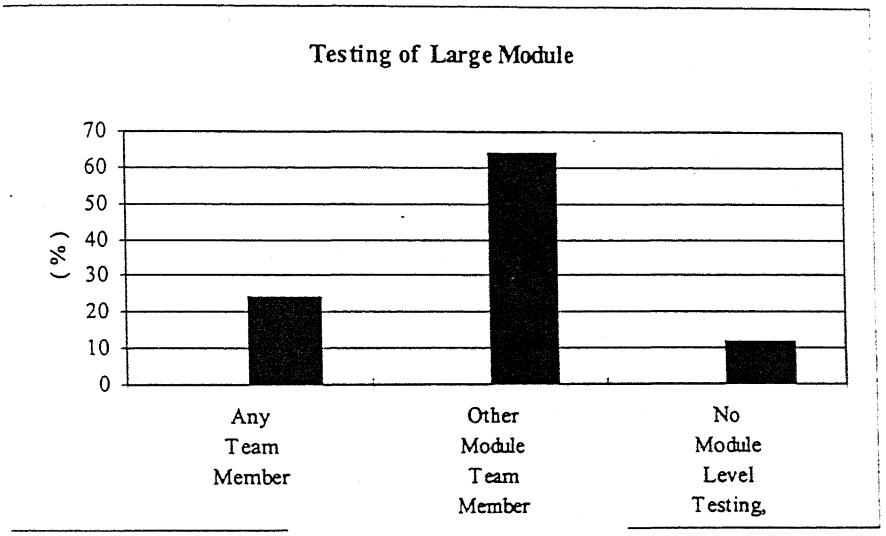


Fig 4.8.2

4.8.1 *Discussion*

From the statistical analysis it is evident that the specialised tester function does not exist in the module team as proposed in the Surgical team concept (refer table 4.6 of appendix1 of the chapter). Also Module Leader or Project Leader is not expected to carry out the testing of module as proposed in Chief Programmer Team paradigm. In case of small modules any team member or member of other team is expected to carry out this function.

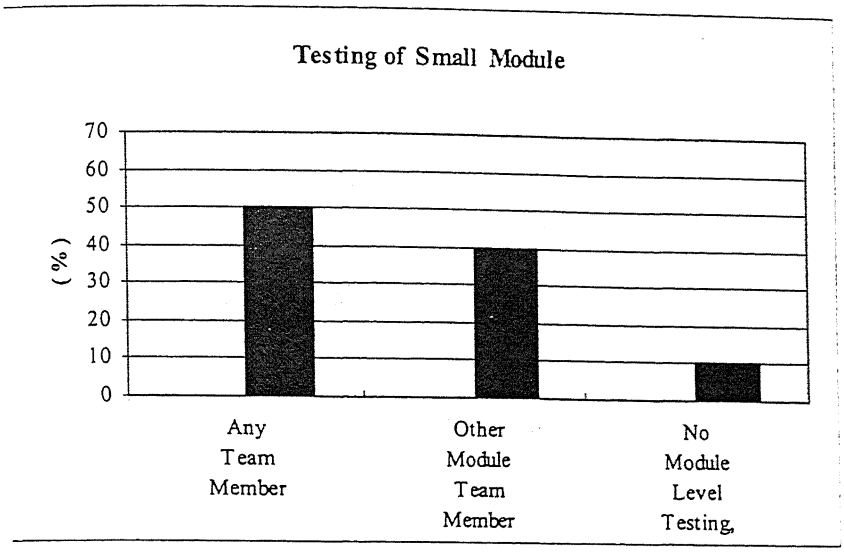


Fig 4.8.1

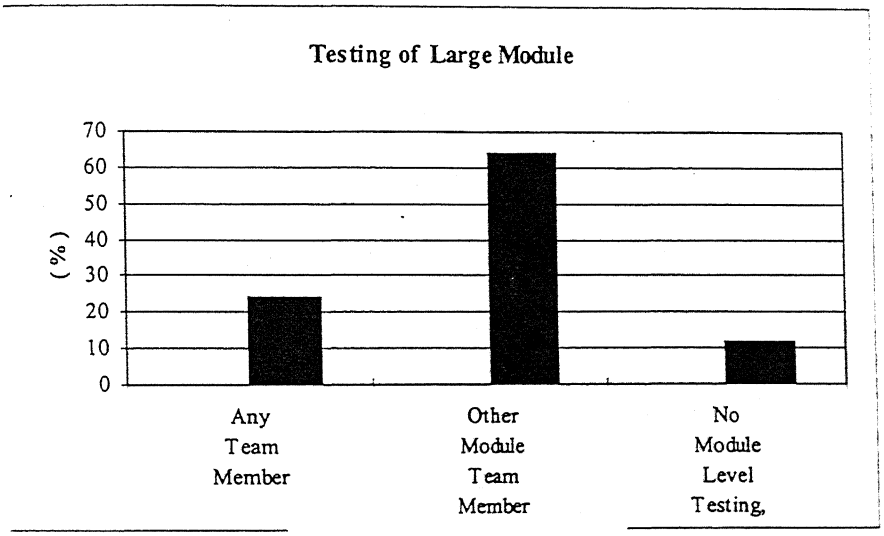


Fig 4.8.2

4.8.1 *Discussion*

From the statistical analysis it is evident that the specialised tester function does not exist in the module team as proposed in the Surgical team concept (refer table 4.6 of appendix1 of the chapter). Also Module Leader or Project Leader is not expected to carry out the testing of module as proposed in Chief Programmer Team paradigm. In case of small modules any team member or member of other team is expected to carry out this function.

In case of large modules other team members are expected to carry testing function in majority of cases. Only in very few cases module level testing is not done. It is tested only after integrated with other modules for making of product. Since the large modules are more complex and critical in comparison to small modules, the testing is carried by other team members in addition to testing by any team member.

4.9 *INTERFACE MANAGEMENT*

Interface Management concerned with the structure of software module and position of Programs in it. Question number 20 focuses on this aspect of software development. The Programming Clerk is expected to have knowledge about total structure of module and make all the computer runs visible to all team members. It helps to transform programming from private art to public practice and helps in identifying all programs, data as team property, not private property. According to Chief Programmer Team and Surgical Team concept , it is a full time job of a person. He is expected to do clerical job in the team. But in actual practice Module Leader or Project Leader is expected to take this responsibility. The type and size of software have no influence on this function (Refer table 4.10 of appendix1 of the chapter). Only in two cases it was a full time job of a programmer where the module was very complex and critical. Since sixteen of the twenty five respondent companies were very young with employee strength less than 40, they are unable to take very big and complex projects. So the Interface Management is not a full time job for a team member. Module Leader and Project Leader are concerned with this function.

4.10 *CO-ORDINATION AND COMMUNICATION*

Since the development of software is a team activity, there has to be a lct of co-ordination and communication between members of same team and with other team to maintain integrity of software product. Question no 8,22, 23, 24, 25 & 26 attempts to find out answer to the problem of co-ordination and communication required in the team.

4.10.1 *Communication Inside the Team*

From response of question number 22 of questionnaire, it was found that there is a lot of informal communication in the team. The type of software and size of team have no effect on communication pattern in the team as every respondent felt that informal communication is the main mode of communication in case of small size software teams and formal meetings in addition to informal communication are the main mode in large sized software teams. In no case formal communication like passing memo, paper, documents was found in the team. In some cases the members found communicating through e-mail. While inquired about the number of meetings conducted inside the team in different stages of development through question number 23, a wide range of answers were received for design stage. It ranges from once in a week to daily. It is also independent of type of software. The intensity of communication is less during the implementation stage in comparison to design stage. It depends on the requirement of project. From response to the question number 24 it was found that in small module teams a member presides over the meeting if there is absence of module leader and this role is rotated in the team. This type of working style resembles to concept of egoless programming team. If the project is very small and consists of only few modules, then project leader takes the job of facilitator in the team. From response to question number

25, it was found that overall 60% of teams have an Information Manager, who notes down the issues discussed in the team. Because lot of effort is consumed in communication between team members. This is a part time job of a programmer in the team and this role is rotated among the team members. From question number 26 it was found that in every types of software the programmers get the code written by them checked by other team members before submitting. This helps to ensure integrity of team and ensuring egoless programming concept. Only in five cases of System software, the Module Leader or Project Leader was expected to read every line of code written by other programmers. So that the whole project is visible to him. It is due to greater complexity of System software.

4.10.2 *COMMUNICATION WITH OUTSIDE TEAM*

Question number 8 of the questionnaire deals with communication with other teams and with the user of the product. The communication with other teams of the same project is taken care by Module Leader or Project Leader depending upon the size of the project. The concept of Co-Leader is non-existent in the module team to take care of co-ordination and communication work with other teams and user of product. According to the ideal structure, he is expected to help Module leader in designing the structure of module. It was only found in three cases where the project size was very large. For 46% of cases where there is absence of formal Module Leader, every team member of small module is expected to arrange the meeting between various teams to maintain co-ordination with other teams which is facilitated by Project Leader. In small projects Project Leader is expected to take this responsibility. In case of large modules the Module Leader takes this responsibility. Project Leader discharges this function when the project consists of few modules. Fig 4.10.1 shows different ways in which the responsibility is shared. From the

figure it is evident that all team members of small team participate in communication with other teams more often than large team. The probability for this is 46% for small software teams in comparison to 19% in large software teams.

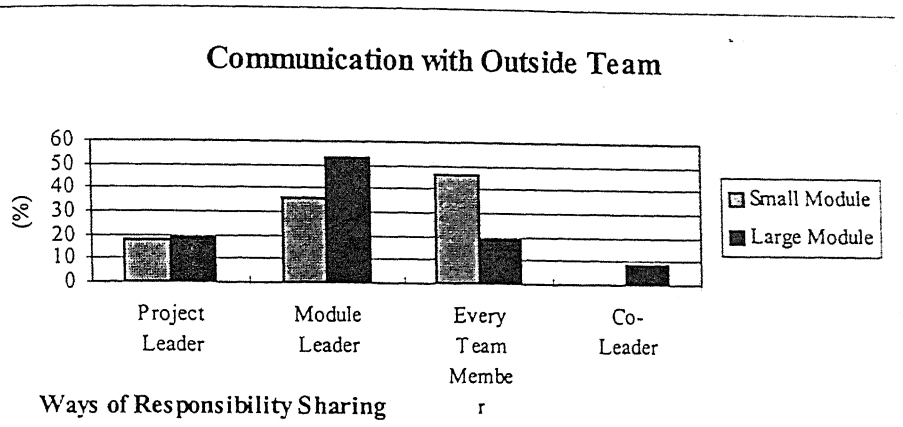


Fig 4.10.1

According to the respondents, communication with user is not done at the module level unless the project is very big and complex. Usually Project Manager takes this responsibility. Since the companies visited were of small size and young, they were unable to take very big projects. So in no case communication of team directly with customer was found.

4.11 *SOFTWARE TOOLS*

Software tools are integral part of development of software. These are used by programmer as aid in development work. Question number 15 of questionnaire tries to find out how knowledge of tools is accessed by the team for different types of software. It was found out that in 52% of System software modules, all the members are expected to have good knowledge of tools in comparison to 28% in case of Application software.

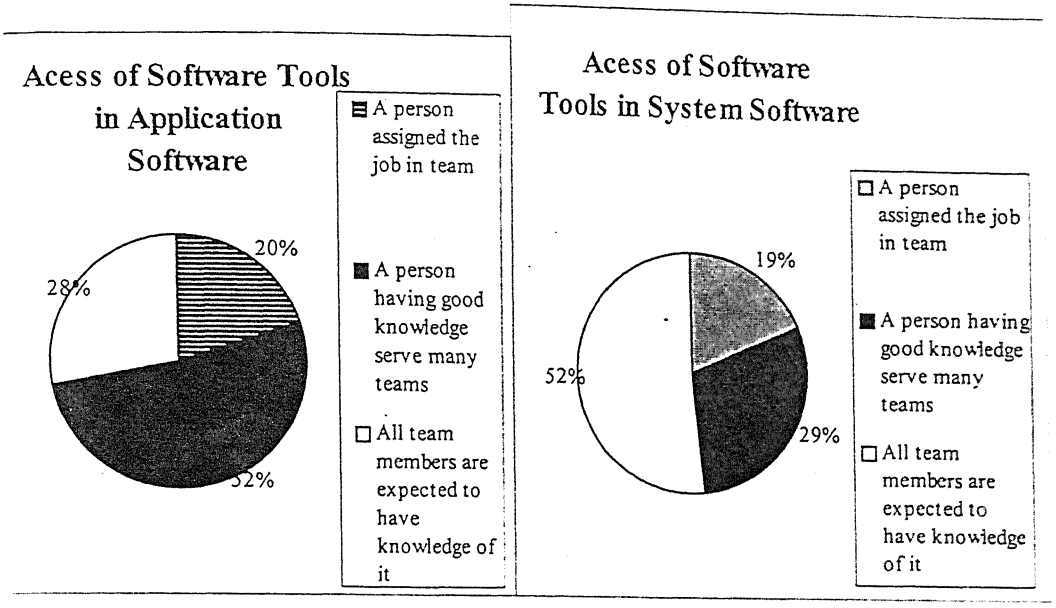


Fig 4.11.1

The above pie chart shows the difference between Application software and System software in accessing different software tools. From the figure it is apparent that in Application software the members of team are not expected to have deep knowledge about the software tools.(Refer Table 4.7 of appendix 1 of chapter) Generally a person having good knowledge about tools serves different teams. In 19 percent of cases a member of team having good knowledge about tools discharges this responsibility. Since in 85% of cases the programmers of System software were having degrees in computer science, they were expected to have good knowledge about the tools. So in 52 percent of cases the respondents felt that every member of team should have good knowledge about the tools.

4.12 *MAINTENANCE WORK*

Maintenance work takes about 67 percent of total effort and resources for a typical heavy duty software (Gunther, 1985). So maintenance work carries a lot of importance and requires a lot of ingenuity. Question no 27,28 & 29 of questionnaire address this aspect of software life cycle. No difference was observed between Application and System software in addressing the maintenance problem. According to respondents , addressing of maintenance problem depends on the organisational variable rather than type of software and size of team (refer table 4.11 of Appendix1 of the chapter). It was found that same development team takes charge of maintenance work for small modules and separate team for large module in seven cases. The reasons for same development team doing maintenance as pointed out by respondents are

1. The work is not big enough for two teams.
2. The co-ordination and communication effort between development team and maintenance team exceed comparing to the amount of work.

The main reason for different teams for both phases of life cycle is due to

1. The management can give more attention to maintenance work to fulfill users demand without disturbing the development work.
2. Clear accountability for both maintenance expense and investment costs of new development.
3. Buffering of new system development personnel from intermittent demands of maintenance.
4. Supports a focus on improved level of service to the user, by means of maintenance specialization.

5. Increased productivity in maintenance, through concentration of system familiarity.

But this type of division creates increased cost of co-ordination between development and maintenance units, especially where new systems are replacement systems. Usually the Module Leader of development team takes charge as leader of maintenance team or the maintenance team works separately under the project leader of development team to get rid of communication problem. The Project Leader or Module Leader works as a communication channel between development and maintenance team. To further facilitate the maintenance work, there is exchange of programmer between development team and maintenance team. By exchanging programmer, the developer could understand maintenance problems and maintenance people could understand the user difficulty better to modify it in the development phase. This also helps to solve status difference between development and maintenance team. Because the maintenance work is seen as lower grade work in industry as comparison to development work. Whenever there is no provision of exchange of programmer, the development team participates in the designing of structure of the software, so that they can check the quality, and can suggest ways to make the software easy to maintain. Generally the maintenance people control the quality of software which makes their work easier in later phase of life cycle of software.

The appendix 1 of this chapter gives various tables of the collected data. The next chapter tries to find out the overall big picture of small software module team, large software module team and compare it with different ideal software team structures discussed in chapter 2 of this writing.

4.13 APPENDIX 1 - *TABLES OF DATA COLLECTED*

Table 4. 1 - EMPLOYEE STRENGTH OF THE SAMPLE ORGANISATIONS

Employee Strength	% of Organisation
< 40	64
40 - 200	20
> 200	16

Table 4. 2 MODULE TEAMS HAVING MODULE LEADER FOR DIFFERENT TYPES OF SOFTWARE TEAM

Tupes of Software Module Team	% Having Module Leader
Small System Software Module	58%
Small Application software Module	45
Large System Software Module	80
Large Application Software Module	85

STATISTICAL TESTING

Types of Software Team	Chi-Square Value (Calculated, Table)	Conclusion
Small Vs Large	(10.178, 2.71)	Reject H_0
System Vs Application	(0.85, 2.71)	Accept H_0

Table 4. 3 - QUALIFICATION OF MODULE LEADERS FOR DIFFERENT TYPES OF SOFTWARE MODULE

Qualification / Types of Team	B.E, B.Tech, M.Tech (Other than Computer Science) % age	B.E , B.Tech, M.Tech (Computer Science) % age	Msc, MCA % age
System Software	12	83	5
Application Software	52	20	28
Small Team	30	53	17
Large Team	32	51	17

STATISTICAL TESTING

Types of Software Team	Chi-Square Value(Calculated, Table)	Conclusion
Small Vs Large	(1.05, 2.71)	Reject H_0
System Vs Application	(9.68 , 2.71)	Accept H_0

Table 4. 4 - NUMBER OF YEARS OF EXPERIENCE OF MODULE LEADER

No of Year Experience / Types of Software	1 - 3 Years (%)	3 - 5 Years (%)	> 5 Years (%)
System Software	35	55	10
Application Software	60	35	5
Small Team	49	44	9
Large Team	47	46	9

STATISTICAL TESTING

Types of Software Team	Chi-Square Value(Calculated, Table)	Conclusion
Small Vs Large	(0.97, 2.71)	Accept H_0
System Vs Application	(0.7 , 2.71)	Accept H_0

Table 4. 5 - DIFFERENT WAYS OF ADMINISTRATIVE DECISION MAKING IN SOFTWARE TEAM

Ways of Decision Making / Types of Team	P.L + M.L or P.L only(%)	Top Management + P.L (%)	Professional Manager (%)
System Software	37	50	13
Application Software	31	57	12
Small Team	16	84	0
Large Team	59	26	15

STATISTICAL TESTING

Types of Software Team	Chi-Square Value(Calculated, Table)	Conclusion
Small Vs Large	(8.97, 4.6)	Reject H_0
System Vs Application	(0.77 , 4.6)	Accept H_0

Table 4. 6 - WAYS OF CARRYING OUT TESTING FUNCTION IN DIFFERENT MODULES

Types of Software Team	Any Team Member (%)	Other Team Member (%)	Integration Testing (%)
System Software	39	51	10
Application Software	37	53	10
Small Team	50	40	10
Large Team	25	65	10

STATISTICAL TESTING

Types of Software Team	Chi-Square Value(Calculated, Table)	Conclusion
Small Vs Large	(8.78, 4.6)	Reject H_0
System Vs Application	(0.94 , 4.6)	Accept H_0

Table 4. 7 - WAYS OF ACCESSING SOFTWARE TOOLS

Types of Software Team	Every Member should have some knowledge of it (%)	Accessed from outside the team (%)
Small Software Team	39	61
Large Software Team	31	69
Application Software	28	72
System Software	52	48

STATISTICAL TESTING

Types of Software Team	Chi-Square Value(Calculated, Table)	Conclusion
Small Vs Large	(8.78, 2.7)	Reject H_0
System Vs Application	(0.94 , 2.7)	Accept H_0

Table 4. 8 - PRESENCE OF SPECIALIST IN THE TEAM

Type of Software Team	Presence of Specialist of application area in Team(%)
System Software	21
Application Software	32
Large Team	38
Small Team	13

STATISTICAL TESTING

Types of Software Team	Chi-Square Value(Calculated, Table)	Conclusion
Small Vs Large	(5.6, 2.7)	Reject H_0
System Vs Application	(1.97 , 2.7)	Accept H_0

Table 4. 9 - NO OF CASES TECHNICAL WRITER DO CODING WORK

Type of Software	Technical Writer has to do Coding Work (%)
System Sofware	38
Application Software	37
Small Team	63
Large Team	8

STATISTICAL TESTING

Types of Software Team	Chi-Square Value(Calculated, Table)	Conclusion
Small Vs Large	(8.23, 2.71)	Reject H_0
System Vs Application	(0.94 , 2.7)	Accept H_0

Table 4. 10 - PRESENCE OF PROGRAMMING CLERK IN THE TEAM

Type of Software Team	Presence of Information Manager (%)
System Software	15
Application Software	17
Small Module	14
Large Module	18

STATISTICAL TESTING

Types of Software Team	Chi-Square Value(Calculated, Table)	Conclusion
Small Vs Large	(1.23, 2.71)	Accept H_0
System Vs Application	(1.09, 2.71)	Accept H_0

TABLE 4. 11 - MAINTENANCE WORK

Type of Software	Different Team for Development and Maintenance Work (%)
System Software	78
Application Software	73
Small Team	56
Large Team	83

STATISTICAL TESTING

Types of Software Team	Chi-Square Value(Calculated, Table)	Conclusion
Small Vs Large	(2.68, 2.71)	Accept H_0
System Vs Application	(1.13, 2.71)	Accept H_0

TABLE 4.12 - DESIGN OF MODULE STRUCTURE

STATISTICAL TESTING OF PARTICIPATION INDEX

Types of Software Team	Z - Value(Calculated, Table)	Conclusion
Small Vs Large	(2.342, 1.79)	Reject H_0
System Vs Application	(2.736, 1.64)	Reject H_0

From the statistical testing it can be concluded that the Participation Index of programmers of System software higher is than programmers of Application software. Also it can be concluded that the Participation Index of Small software module team programmers are higher than that of Large software module team.

5. FINDINGS OF THE RESEARCH

5.1 *INTRODUCTION*

This chapter frames a big picture of various types of software teams differentiated by size and type of software, mainly System software & Application software. The team structures are compared with various ideal structures discussed in chapter 2 of this writing. Since from the previous chapter we found out that the difference between Application software module team and System software team is not significant, we will discuss about the differences between small teams and large teams and point out the differences between Application software and System software intermittently between the discussion, whenever required.

5.2 *COMPARISON OF SMALL & LARGE SOFTWARE TEAMS*

Two types of structure were found common amongest the small software module teams. The two types of structures resembles to that of Egoless Programming Team and Chief Programmer Team (see section 5.3 for details) with some variations. The division of labour is not distinct in case of small software teams. One type of team structure emerges for Large software team from the analysis. The type of software has very insignificant effect on the team structure. This section shows differences between the Large and Small teams for each step of software development process.

5.2.1 *Formal Leadership And Team Management*

5.2.1.1 SMALL SOFTWARE MODULE TEAM

The formal Module Leader was found in about 50% of the cases in Small teams. There is no difference between Project Leader and Module Leader for the project which consists only of a single module. The Project Leader is expected to take administrative decision with the help of Top Management and guide the team. Whenever there is presence of Module Leader, Project Leader carries out the administrative function in consultation with him/her. Only in small organisations where the employee strength is less than 40, Top Management carries out the administrative function and Module Leader is not expected to be responsible for this function.

5.2.1.2 LARGE SOFTWARE MODULE TEAM

The formal Leadership exist for more than 80% of the cases. Except for the project which consists of only one module, Module Leader is present in every software module. Only in very small organisations where the employee strength is less than 20, the formal leadership does not exist in the team. In that case top management acts as the leader of the team. Like that of Small teams, Module Leader is not expected to carry out the administrative responsibility of the team. The Project Leader takes the administrative decision in consultation with the Module Leader. In small organisations, the top management takes up the administration of the team. Only in very large organisations where the total employee strength exceeds 200, a member of team is an administrator

who acts as a communication channel between top management and the team. So specialised administrator function exist only for very large organisations.

5.2.1.3 COMPARISON WITH IDEAL STRUCTURES

The specialised administrator function does not exist in the Small software module team as suggested in Chief Programmer team and Surgical team. It only exists for Large teams in very big organisations. Also this decision is not reached through general consensus as suggested in Egoless Programming team. The Project Leader and Top Management are expected to carry this function in consultation with Module Leader if he/she is present in the Small software module team. The administrative decision is a collective decision of Project Leader and Module Leader in Large teams.

5.2.2 *Design Of Module Structure*

5.2.2.1 SMALL SOFTWARE MODULE TEAM

The Project Leader and Module Leader designs the module structure with help of the team members. Whenever there is absence of Module Leader, Project Leader designs the module structure with the help of team members. It seems that the entire team is given a target or goal instead of assigning the individual responsibility for this particular function of software development. The technical difficulty of the Application area is solved by the Project Leader or Module Leader. In the complex and critical modules, there is presence specialist of application area in the team. In Small modules the participation of programmers is higher in comparison to Large module teams. The distinction between

the designer and implementor is very low in case of Small software module as the participation index of programmers in small module is very high. The team members participate in the design process according to their capabilities. The Participation Index of members of the System software modules is higher than that of Application software. Barring four organisations, it was found that there is no difference between Participation Index of programmers for different types of software in a single organisation. Because the qualification of programmer does not vary in an organisation. For the organisations which are making only System software, the Participation Index was found higher in comparison to organisations developing only Application software. It was because of higher qualification and expertise of programmers in System software.

5.2.2.2 LARGE SOFTWARE MODULE TEAM

The difference between designer and implementor is more distinct in case of Large software module in comparison to Small module as the Participation Index of programmers is lower. The Module Leader does the design of the module in consultation with Project Leader or team members. The difference between System software and Application software is same for Large modules as that of Small modules. The probability of presence of Specialist on application area is more than in Small modules.

5.2.2.3 COMPARISON WITH IDEAL STRUCTURES

It is evident from the analysis that the Participation Index of programmers is not zero In both Small and Large module teams as suggested in Chief Programmer team and Surgical

team. Also not all the members are intensely involved as suggested in Egoless Programming team. The participation of members depends on their capabilities and qualifications. The design of module structure is collective responsibility of all the team members. The major deviation of practical team structure from ideal structure is in terms of having a specialist in the application area concerned. According to ideal structures, the Module Leader should be a specialist of application area of software module. But in general practice he acts as a generalist in the team having some idea about application area, the programming environment and user needs etc. rather than a specialist of the application area in both Small and Large software module teams.

5.2.3 *Technical Writing*

The technical writing or editing is a specialised function of a member in the team to prepare the external documentation of the software product. He is not expected to write code in both Small as well as in Large module teams. Only in few cases of Small software module team where the work is not substantial, he is expected to write some code. Also if the work is not substantial for full time job of a member in the team, then one technical writer may serve more than one teams. All the members are required to do the internal documentation of their work to maintain clarity of work. It also helps to understand the code in case the programmer leaves the organisation.

This characteristic of module team resembles Surgical team structure with having a technical writer in the team unlike Chief Programmer team, where the Chief Programmer

is expected to fulfill this responsibility or Egoless Programming team, where this work is shared by all the members equally among themselves. It is due to importance of technical writing for better user understanding and clarity of software product. The size of team has no influence on this aspect of software development.

5.2.4 *Testing of Software Module*

The members of Small software module team are expected to carry out the integration and testing work with the guidance and help of Project Leader. If the module is very complex and critical, then other team members are called to carry out the testing function of the module.

Testing by members of other team is the main mode of testing in case of Large software module teams. Only in 25% of the cases any team member is expected to carry this function required for software development.

There is no specialised Tester function in the team as suggested in Surgical team structure. Also the Module Leader is not expected to test every line of program as suggested in Chief Programmer team in both Small as well as Large software module teams.

5.2.5 *Interface Management*

The size of team has no influence on this aspect of software development. The Interface Management function is shared by all the team members in supervision of Module Leader or Project Leader. If the module is very large one member of the team is expected to keep the programs, but he is not expected to carry out the integration work as suggested in Surgical team. Also Module Leader is not expected to carry out this responsibility as suggested in Chief Programmer team. The responsibility of Program Clerk of keeping code visible throughout the team does not exist in the team. Instead the programmers exchange code among themselves to promote Egoless programming paradigm and to keep the code visible throughout the team.

5.2.6 *Software Tools*

The specialised toolsmith function does not exist in the team as proposed in Surgical team paradigm. The size of team has no influence on this aspect. In case of System software, the team members are expected to have some knowledge of software tools and a more knowledgeable person solve their difficulty. In Application software the help of specialist from outside the team is more often taken.

5.2.7 *Co-ordination & Communication*

5.2.7.1 SMALL SOFTWARE MODULE TEAM

The co-ordination and communication with other teams and user is shared by all the team members in the absence of Module Leader. All the team members participate in meeting with other teams presided over by Project Leader. If the project is very small, then Project Leader takes this responsibility alone. If there is presence of Module Leader, then he is expected to carry out this function required for software development.

The main mode of communication between team members is informal communication in the design as well as implementation stage of software development. In no case the formal communication by passing memo, paper etc. was found in the team. It provides efficiency and results in smooth operation. If there is absence of Module Leader, then the facilitator role in meetings is played by a programmer and this role is rotated. In some cases Project Leader takes this responsibility. If formal Module Leader is present in the team, then he is expected to play the role of facilitator in the team. The information manager role is played by a programmer in the team. He is expected to note down the issues discussed in the team. Because a lot of effort is consumed in communication between team members. This role of information manager is rotated among the team members. The programmers get the code written by them checked by other team members before submitting. This helps to ensure integrity of team similar to Egoless Programming concept.

5.2.7.2 LARGE SOFTWARE MODULE TEAM

In large team the communication with other teams and users is taken by Module Leader and Project Leader. All the team members are not expected to communicate directly with user and members of other team in Large software module team unlike Small module team where this responsibility is shared by the programmers where there is absence of Module Leader. The information manager role is played by a programmer in the team. This role is rotated among the team members. Like that of Small module teams the programmer exchange code among themselves to cultivate egoless programming environment.

Like that of Small software module team, the main mode of communication in the team is informal communication. In addition to informal communication, formal meetings are conducted in the team for conveying ideas from Designer to Implementor. Module Leader of the team acts as facilitator in the team meetings.

5.2.7.3 COMPARISON WITH IDEAL STRUCTURES

According to both Surgical team and Chief Programmer team paradigm, there should be a Co-leader in the team to communicate with user and other teams of the same project. But in no case of Small software module and Large software module, the co-ordination and communication with user and other teams is a specialised function of the team

member. The communication with user is generally done at Project Manager level. He is responsible for communicating user needs to the team.

5.3 *DISCUSSION*

This section discusses the structure of software module teams in actual practice. The structures are then compared with the ideal structures to find differences from them.

5.3.1 *Structure of Small Module Team*

Two types of team structures are visible for Small software teams. The first type of team structure resembles with Egoless Programming team structure having little more division of labour than ideal team structure. There is presence of Project Leader, Technical Writer and Specialist of application area depending upon the criticality of software module. To overcome the shortcomings of the Egoless Programming team in communication with user, other teams and in administrative work, the Project Leader provides necessary guidance. He works as an interface between Top Management and the team. The members contribute towards achieving the goal according to their capabilities rather than assigning particular responsibility to an individual member. The main mode of communication between the team members is informal communication. The priorities lie in harmony, mutual identification and effortlessness in co-ordination, where the goal is achieved by the efficient harmonious alignment.

The second type of Small software team structure resembles with Chief Programmer team structure with little variation to that of ideal structure. The characteristics of structure of team are

1. In this type of small module team there is a presence of formal leader who is called Module Leader. He is expected to design the module structure with the help of Project Leader and team members depending on their capability. But he is not expected to write code of critical portions of software as suggested in the Chief Programmer team structure. He is also not expected to test every line of code as that of ideal team structure. He may or may not write code according to the requirement.
2. The testing function is done by any team member or other team members according to criticality and complexity of the module. Testing function is not a specialised function of a team member as suggested in Surgical team structure.
3. There is absence of Program Librarian in the module team. The integration work is expected to be carried out by the module leader with help of team members. Every programmer gets her/his code checked by other team members before submitting to promote the concept of Egoless Programming.
4. Technical writing is a full time job of a member in the team. It is a major variation from the Chief Programmer team paradigm where the Chief Programmer is expected to carry out this responsibility in ideal structure.

5. In most of the cases the Project Leader and Module Leader is expected to be an expert in the application domain. All types of structures do not provide a specialist of application area other than Module Leader in the team. But in actual practice a specialist of application area may be a member of team and helps Project Leader and Module Leader in design work depending upon the complexity and criticality of the project. He is expected to sort out any difficulties in design concerned with application domain.

6. In no case an administrator was found in the team. It is not a specialised function of a team member. The project leader is expected to carryout the administrative functions in the team in consultation with Top Management.

7. User Liasion function also does not exist in the team as suggested in the Revised Chief Programmer Team concept. This is because of the fact that projects taken by all these companies were not very big. So the communication with user does not come to the module level.

The team structure of these types of small team resembles to that of stuctured open team (Refer Appendix 5) with certain deviations from ideal structure. The responsibilities are not clearly defined as that of Structured Open team. There is lot of informal

communication in the team among the members. Each team member participates in the team activities according to their capability. In many cases there is presence of information manager and facilitator as explained by Constantine in his article "Orchestrating Project Organisation & Management" published in Oct 1993 issue of 'Communication of the ACM'. The information manager's role is played by any programmer and this role is rotated among them to promote team sprit and motivation. So the team functions like a structured open team with less division of labour than ideal structure.

5.3.2 *Structure of Large Software Team*

Egoless Programming structure is absent for this type of module structure. The structure resembles to that of Chief Programmer Team with some more division of labour. The division of labour is not distinct as that of Surgical team. In no case the division of labour resembles to that of Revised Chief Programmer team.

1. In every Large software module, there is presence of a formal leader who is Module Leader. He acts as a generalist rather than an expert of application area as suggested in Surgical team and Chief Programmer team. The probability of having Specialist of application area is more in comparison to Small software module teams. The specialist administrator function does not exist in the team except for very large organisations. The Project Leader takes this decision in consultation with Module Leader and top management. The Module Leader is not expected to write code unlike Small software module teams where he may write code according to the requirement. But he need not be a super

programmer as suggested in Chief Programmer team. Also he need not read every line of code as that of ideal structure. The Co-leader does not exist in the team to act as a communication channel between team , other team of same project and user of the product. This role is played by Module Leader.

2. The specialised tester function does not exist in the module team. Other team members are required to carry out the testing function. In other cases any team member may carry out the testing function.

3. Technical writing is a specialised function of a team member. It is a major variation from ideal Chief Programmer team where Module Leader is expected to carry out this responsibility.

4. The integration work is done by team members and Module Leader depending upon the work. The specialised Programming clerk function is absent as that of small software module. To make the software development process visible to all team members, the programmers exchange code among themselves.

The communication pattern in the team is mostly informal. In addition to informal meeting between members, formal meetings are conducted in the team. In most of the cases a particular day of week is assigned for meetings. The role of facilitator is played by Module Leader of the team. The information manager is present in every Large software

module team. He writes down the issues discussed in the meeting in a systematic manner, so that the team does not lose track of the issues discussed. This role is played by a programmer in the team and it is rotated. The Large software module team works like the Structured Open team with certain variations. The Module Leader has the final say about the outcome of the issues discussed in the meetings in most of the large modules. It brings discipline and saves the proceeding from chaotic enmeshment and endless processing of discussed issue which is a characteristics of Structured Open team (Refer Appendix 5). The members share information in an informal atmosphere. All the members give the combined feedback and responses on every issue of software development. The member's contribution depends on their capability and qualification. It creates a good learning environment for every team member. But the roles of team members are not defined distinctly as should be in a Structure Open team.

The table 5.1 in next page compares the difference between Small team and Large team for each step of development process. The table shows the difference between two types of Small software module team and Large module team. Table 5.2 gives division of labour for each steps of software development in the four ideal structures.

The next section of this chapter describes some possible cause of deviation of actual team structures from ideal structures.

TABLE 5.1 - Division of Labour in Small and Large Module Team

NOTATION USED

M.L - Module Leader

P.L - Project Leader

T.M - Team Members

TYPES OF MODULE TEAM / STEPS OF DEVELOPMENT	SMALL SOFTWARE MODULE TEAM STRUCTURE		LARGE SOFTWARE MODULE TEAM
	TYPE I	TYPE II	
Formal Leader	No Module Leader	Module Leader	Module Leader
Administrative Work	P.L	M.L + P.L	P.L + M.L and administrator for big organisation
Design Work	P.L + Team members according to capability	P.L + M.L or P.L + M.L + T.M according to their capability	P.L + M.L or P.L + M.L + T.M according to their capability
Technical Writing	Specialised job of a programmer	Specialised job of a programmer	Specialised job of a programmer
Testing	Any member / Other team members	Any member/ Other team members	Predominantly other team members
Interface Management	Combined work of all team members	Work of programmer in guidance of M.L	Work of programmer in guidance of M.L
Coding Work	All members	Programmers + M.L (If Necessary)	Programmers
Access of Software Tools	No Specific Pattern	No Specific Pattern	No Specific Pattern

Communication pattern in team	Informal Communication	Informal Communication	Informal Commu. + Formal Meetings
Communication with outside team	Shared by all the members / P.L	Responsibility of Module Leader	Responsibility of the Module Leader
Information Management	Carried out by a programmer and it is rotated	Carried out by a programmer and it is rotated	Carried out by a programmer and it is rotated
Facilitator of the Meetings	Any Member / P.L	M.L	M.L

TABLE 5.2 - Division of Labour in Ideal Team Structures

Notations Used

E.P.T - Egoless Programming Team

S.T - Surgical Team

C.P.T - Chief Programmer Team

R.C.P.T - Revised Chief Programmer Team

M.L - Module Leader

C.L - Co-Leader

STEPS OF SOFTWARE DEVELOPMENT	**E.P.T**	**C.P.T**	**S.T**	**R.C.P.T**
Formal Leader	No Leader	Module Leader	Module Leader	Administrator
Administra. Work	All Members	Administrator	Administrator	Administrator
Design	All Members	M.L + C.L	M.L + C.L	M.L + C.L
Technical Writing	All Members	M.L	Editor	Editor
Testing	All Members	M.L	Tester (Specialised function)	Programmer + C.L + M.L
Interface Management	All Members	Programming Clerk	Programming Clerk	Programming Clerk
Coding	All Members	Programmers + M.L(critical portions)	Programmers	Programmers
Communication with other teams	All Members	C.L	C.L	C.L
Comm. with User	All Members	C.L	C.L	User Liasion
Decision making body in all cases	All Members	M.L	M.L	Administrator

5.4 *THE CAUSE OF DEVIATION FROM IDEAL TEAM STRUCTURES*

It was found out that the actual team structure varies from ideal structures. The Surgical team structure is not practicable in actual application. The various reasons for the deviations may be as follows :

1 All these structures are proposed for making development process visible to management for better control and to appraise the performance of individual members. It is necessary only when the organisation is very big and management has to oversee the development process of a large number of softwares. But the companies visited were very small organisations having employee strength of less than 40 in sixteen out of 25 cases. The number of projects taken by them is very few. So the management can oversee the development process of all projects undertaken. In all these organisations management is directly involved in the development process. So the team is structured to make production process smooth rather than management control.

2 The ideal team structures are proposed by individuals associated with development of very large projects of some thousand man years. All the authors of different ideal software teams were directly or indirectly associated with IBM Corporation of USA, which generally develops very big software. For example, the Surgical team is proposed by Federic Brooks, who was associated

with development of system IBM/360 and was one of the chief designer. It was a project of 5000 man years. Nearly 1000 people were working on it. As the communication effort increases exponentially with the number of people, the work has to be divided very systematically for better management control. Operating system IBM 360 is one of the most complex systems developed till now. But the companies visited were working on projects much smaller in comparison to Operating System IBM/360. In all the cases the maximum number of people working in a project is around 35. In some cases the total project consists of a single module where the number of people working is less than 10. So the extensive division of labour only creates hindrance in the smooth process of development. Also the many steps like Testing, Interface Management, Toolsmith function is not a specialised function of a single person in the team. These responsibilities are fulfilled by Module Leader. Most of the Indian software companies are developing parts of big project for some other developer. So they are doing very low value added work like coding without bothering about the design and interface management aspect of the project. So the specialised functions for all these aspects of development is missing in the team structure

The next section discusses about the learning from ideal structures.

5.5 *LEARNINGS FROM THE STUDY*

After finding out the actual team structures and comparing with ideal structures, we came to the following learnings :

1. The actual team structures have very less division of labour in comparison to ideal structures. The ideal structures are only suitable for very big organisations developing very large software product.
2. The team structure resembles to that of Chief Programmer team structure with some variations to ideal structure.. The Surgical team concept is not applicable to the actual team structure due to extensive and complex division of labour which is very difficult to put in practice.
3. The role played by programmers in different stages of software development is determined by their capability, qualification and capability of Module Leader rather than fixed prior to beginning of the development process.
4. Many specialised functions like Tester, Language Lawyer, User Liasion functions do not exist in the team. If the project is very large, then only the communication with user comes to the team level. Technical Writing is only specialised function of a team member.

The next chapter draws conclusion of the study.

Chapter 6

6.CONCLUSION

6.1 *INTRODUCTION*

This chapter gives the conclusion of the study. It also describes various limitations of the study and scope for further work. The objective of thesis was to find out the actual working style of software module teams and compare it with different ideal structures proposed in the literature to find its validity in the Indian context. The study attempted to find out the differences of team structures for different types of software - Application software and System software. The software modules teams were also categorized into Small and Large modules according to the number of people working in it. Ideal structures with which the actual structures are compared were Egoless Programming team, Chief Programmer team, Surgical team and Revised Chief Programmer team. These details and differences are discussed in section 2.3.

The questionnaire survey method was followed for this study. The various steps addressed in the questionnaire were Formal Leadership and Management, Design, Solution of Application Difficulty, Technical Writing, Review of Design, Testing, Interface Management, Co-ordination and Communication, Software Tools and Maintenance work.

From the analysis of data it was found that the type of software of System and Application has no influence on structure of the team. Some differences were observed between Small and Large software module teams. Two types of structures were visible in case of Small software module teams. These two team structures resemble to Egoless Programming team and Chief Programmer team with certain variations. The working style of teams resemble to Structured Open team. The Large software module teams resembles to Chief Programmer team with some variation in division of labour from ideal structure. The Module Leader of team is not over burdened like Chief Programmer and his responsibility exceeds to that prescribed for Chief Surgeon in Surgical team paradigm. The specialised administrator, tester, toolsmith and Programming Clerk functions do not exist in the team as suggested for Surgical team in both Small and Large module teams. The administrator functions exists for Large software teams only in some very big organisations. The testing function is carried out either by any team member or members of other team. It is predominantly by other team members in case of Large software teams. The interface management function of programming clerk is carried out by Module Leader and programmer instead of assigning a member this particular function. Only specialised Technical Writer's function exists in both Small module team and Large module team. Only in some cases of Small teams, the Technical Writer is expected to write some code. The Module Leader is not expected to carry out this responsibility as suggested in Chief Programmer team. The role of Co-leader to act as a communication channel between team, other teams of same project and users of the software product does not exist in the team. This responsibility is either carried out by Module Leader or shared by all the team

members among themselves. The participation of programmers in design depend upon their capability and qualification. The Large software team works like structured open team. But the division of labour and responsibility of programmers are not distinctly defined as that of ideal team.

The next section describes various limitations of this research work.

6.2 *LIMITATIONS*

The various limitations of this study are

- The sample size of the respondent companies is very small to call it a study on Indian Software Industry. Also the respondent companies are situated at one geographic location of the country. So it does not represent uniform mix of various software companies of India.
- Since we were comparing the differences between various types of teams, the organisational variables have not been controlled while showing the differences. The organisation variables has also some effect on team structure as is evident from analysis of the data. The working style teams depends upon various practices adopted in the companies. This factor has not been taken into account while analysing the data.
- Only two responses were collected from each company, which is very small in comparison to total employee strength of the companies. The variation of response among the Programmers and System Analysts in a single organisation has not been taken into account while analysing the data.

- From the analysis of data, it is evident that the division of labour depends on the capability of Module Leader and Programmers. The function carried out by Module Leader depends on her/his capability and qualification. This factor has not been taken into account while analysing the data.

The next section gives further scope of research beyond this study.

6.3 *SCOPE OF FURTHER WORK*

The research work can be further extended to study the effect of capability and qualification of Module Leader on the structure of the team. During the analysis we have shown that team structure varies according to qualification and capability of Module Leader. The case study can be done for two or more teams to show differences between them.

The study can be further extended to find out effects of organisational variables on team structure. The main organisational variables which may be taken into account is type of market they are catering like Software Service or Software Product, employee profile, size, ownership etc. of the organisation etc.

7. APPENDIX

7.1 APPENDIX 1

7.1.1 *What is Different About Software*

The first Appendix is a excerpts Brooks (Brooks, 1975) statements (with some extra material by me) on what is programming all about.

7.1.2 *The Craft, its Woes and its Joys.*

What is being produced by either large industrial teams or garage duos? Brooks categorizes software efforts into four types. He explains with a diagram (see figure) . The four categories are: a program; a programming product; a programming system and the programming systems product. Each type involves different levels of effort and associated cost.

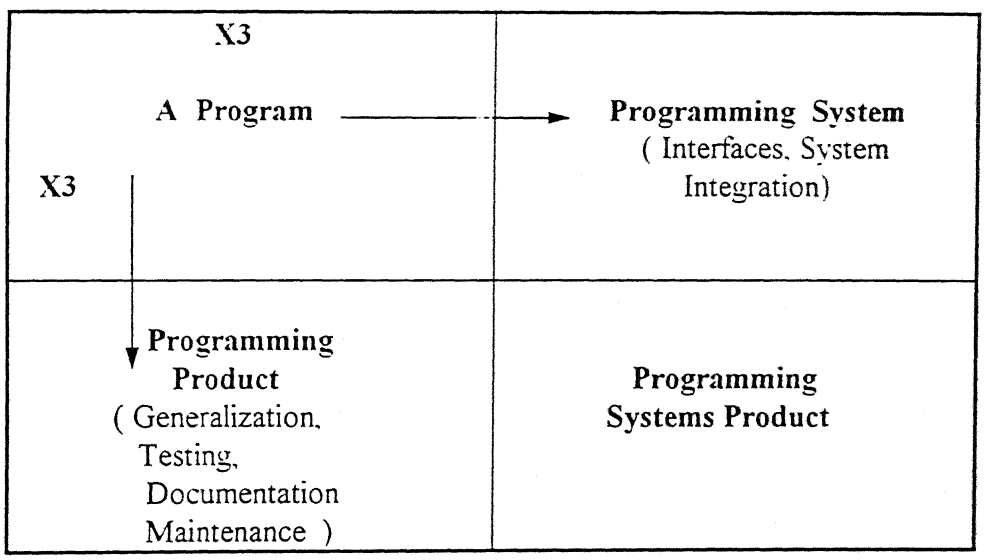


Fig 7.1 The effort required to move from program to product (Brooks, 1975)

A *program,* on the upper left quadrant of the diagram, is complete in itself. It is ready to be run by the author on the system on which it was developed. The most commonly exemplified by the class assignments that student do or the internally developed & used programs of a department. There are two ways a program can be converted into a more useful, but more costly, object. These two ways are represented by the boundaries in the diagram.

Moving down across the horizontal boundary, a program becomes a *programming product.* This is a program that can be run, tested, repaired, and extended by anybody. It is usable in many operating environments, for many sets of data. To become a generally useable programming product, a program must be written in a generalized fashion. In particular, the range and form of inputs must be generalized as much as the basic algorithm will reasonably allow. Then the program must be thoroughly tested, so that it can be depended upon. This means that a substantial bank of test cases, exploring the input range and probing its boundaries, must be prepared, run and recorded. Finally, promotion of a program to a programming product requires its thorough documentation, so that anyone may use it, fix it, and extend it. As a rule of thumb, he estimates that a programming product costs at least three times as much as debugged program with the same function. An smaller scale example would be Wordstar, or Lotus 123, when they first came out. They never became available on other systems, but had documentation, were tested, etc. A fine example are the GNU products like gcc - a compiler, perl etc. that are well documented, run across various platforms. Being public domain, it is extended, fixed by many people across the world, everyday. These kinds of systems are children of

the Internet. Any extensions made, that are found to be useful are incorporated into next version. No matter who made it.

Moving across the vertical boundary, a program becomes a component in *programming system*. This is a collection of interacting programs, coordinated in function and disciplined in format, so that the assemblage constitutes an entire facility for large tasks. To become a programming system component, a program must be written so that every input and output conforms in syntax and semantics with precisely defined interfaces. The program must also be designed that is uses only a prescribed budget of resources-memory spaces, IO devices, computer time. Finally, the program must be tested with other system components in all expected combinations. This testing must be extensive, for the number of cases grows in a combinatorial fashion. It is time consuming, for subtle bugs arise from unexpected interactions of debugged components. A programming system component costs at least 3 times as much as a stand alone program of the same function. The cost may be greater if the system has many components. This can be exemplified by the office suite of programs that are available e.g. MS office, SmartSuite which has Lotus 123 type spreadsheet as just one component of the suite of programs like a database manager, a wordprocessor, etc. The data developed with one of these is components is effortlessly incorporated data of the other component.

In the lower right hand corner of the figure, stand the *programming systems product.* This differs from the simple program in all the above ways. This differs from the simple program in all the above ways. It is generalized, it undergoes testing, has documentation,

is maintainable, is integrated or intergrateable through its well-defined interfaces. It costs 9 times as much. But is the truly useful object, the intended product of most system programming efforts.

After explaining what is it that is made, in rather poetic fashion, Brooks goes on to explain why programming is fun. Later, he goes on to explain why its so much headache.

1. For sheer joy making things. Especially things of own design.
2. For the pleasure of making things that are useful. "Deep within, we want others to use our work and to find it helpful."
3. Third is the fascination of fashioning complex objects of interlocking moving parts, and watching them work in subtle cycle, playing out the consequences of principles built in from the beginning.
4. The joy of always learning, which springs from the non repeating nature of the task. In one way the problem is ever new, and its solver learns something: sometimes practical, sometimes theoretical, and sometimes both.
5. There is the delight of working in such a tractable medium. The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds his castles in the air, from air, creating by exertion of the imagination. Few media of creation are so flexible, so easy to polish and rework, so readily capable of realising grand conceptual structures. This tractability has its own problems. Yet the program, unlike the poet's words, is real in sense that it moves and works machines, controls airplanes, satellites, transfers money between banks etc.

His explanation highlights the non repeatable nature of the task, and the tractability of the medium. He then explains what makes software such a headache.

1. One must perform perfectly. Human beings are not accustomed to being perfect, and few areas of human activity demand it.
2. Other people set would be complete, available and usable.
3. Designing grand concepts is fun; finding nitty little bugs is just work. One finds that debugging has a linear convergence, or worse, where one somehow expects a quadratic sort of approach to the end. So design drags on and on, the last difficult bugs taking more time to find than the first.
4. The last straw is that the product over which one has labored so long appears to be obsolete upon (or before) completion. Already colleagues and competitors are in hot pursuit of new and better ideas. Already the displacement of one's thought child is not only conceived, but scheduled. Of course, the technological base on which one builds is always advancing. As soon as one freezes a design it becomes obsolete in terms of its concepts. But implementation of real products demands phasing and quantizing. The obsolescence of an implementation must be measured against other existing implementations, not against unrealized concepts. The challenge and the mission are to find real solutions to real problems on actual schedules with available resources.

7.2 *APPENDIX 2*

7.2.1 *Software Myths*

The unique nature of software is associated with many software myths. The myths can be broadly divided into three different categories. The different myths are Management Myths, Customer Myths, Practitioner's Myths. All the myths are described by Pressman (1985) in his book on software engineering. This chapter is describes the Management Myths associated with software development.

7.2.1.1 MANAGEMENT MYTHS

Myth1 : The management does not thinks to change their approach to software development. Because same kinds of programming is used as that of languages used ten years ago.

Reality : Although the application domain may be the same (and for many organisations it has changed substantially), the demand for greater productivity and quality and criticality of software to strategic business objectives have increased substantially.

Myth2 : According to management well planned standard procedure like Structured Programming, spiral Model, Top Down Programming exist for smooth development of software.

Reality : The procedure may well exist, but it is never used. Because majority of software professional are not aware of these techniques. Also it does not reflect modern software development practice.

Myth3 : The management thinks that if the development process fall behind schedule, they can add more programmer and catch up.

Reality : Software development is not a mechanistic process like manufacturing. In the word of Brooks (1975): "...adding people to late software project makes it later." At first, this statement may seem counterintuitive. However, as new people are added, the

need for learning and communication among staff can and does reduce the amount of time spent on productive development. People can be added, but only in a planned and well co-ordinated manner.

7.3 *APPENDIX 3*

7.3.1 *Different Stages of Software Development*

The software development process can be divided into five stages like Analysis Phase, Feasibility Phase, Design Phase, Programming Phase, and Evaluation Stage. The functions carried out at different stages are very much overlapped. It is very difficult to separate one stage from other. This chapter gives detail account of various functions in different stages.

7.3.1.1 ANALYSIS PHASE

The Analysis Phase begins when the need for a product is acknowledged by the management. This acknowledgment is usually accompanied by an intent, perhaps tentative or provisional, to develop the product. In analysis phase the energy and attention are focused on understanding the role the proposed product would play. The work that takes place in this phase is the planning and coordinating needed to prepare a formal written set of requirements for the product. The phase ends when you have crystallized requirements to the point where they can be reviewed, modified as necessary, and approved by the management personnel responsible.

7.3.1.2 FEASIBILITY PHASE

The Feasibility Phase is the technical part of the analysis phase. It begins when your management intent has firmed to the point where you authorize and initiate a project and allocate manpower resources to it. The work involved is a study of the proposed product to develop enough understanding to determine a reasonable estimate of feasibility. The

proposed product can be developed or not is determined in this phase. The economic feasibility and marketing feasibility can be is studied in this phase. The feasibility phase ends when the requirements are approved

7.3.1.3 DESIGN PHASE

The Design Phase usually starts early in the feasibility phase, after the analysis and feasibility phase continue for some period of time. When the requirements are approved, the design is in high gear. The design is recorded in the formal specification of the product. The algorithms to be used, the inter-relation between the programs is decided in this phase of the development. The design phase ends when that specification has been formally approved.

7.3.1.4 PROGRAMMING PHASE

In this phase the coding of specifications is done. The programming language is used in this phase to make the specifications into a working of product. It is initiated as early in the design phase as meaningful specifications for portions of the product are available, but not prior to approval of the requirements. The overlapping of programming and design saves a lot of development time. It also serves as a check on the validity of the design and in some cases influences key design decisions. In this phase the work of building the product takes place. It consists of detailed internal design of the software, as well as flow charting, documenting, coding and debugging it. It ends with the start of evaluation stage.

7.3.1.5 EVALUATION PHASE

The Evaluation Phase of the product's life cycle is a buffer zone between the start of integrated testing and beginning of live use of the product. The conformance of the product to user requirement and specification is tested in this phase of development life cycle. It is ensured that the necessary documentation is available, accurately and completely describing the software; and is adequately free of defects. The phase begins as soon as all components are in place and running. It generally lasts as long as the programming phase, but structured programming techniques may reduce it to as little as a third of the programming phase.

7.3.2 *Maintenance Phase*

When the product reaches the final customer to be use, the maintenance phase begins. It lasts as long as the use of product continues. The maintenance phase can also begin at development stage itself for very complex and critical products.

7.4 *APPENDIX 4*

7.4.1 *QUESTIONNAIRE*

1 - Name of the Organization-

2 - The total technical employee strength and the total turnover from the software - [ ] and [ ]

3 - Type of Software made by you (Please tick mark)
I - System Software (Where speed & performance of programs carries a lot of importance)
II - Application Software (Where speed & performance in not very important)
III - Both

* Some definition of terms that will be used in questionnaire below

Small Module - The number of persons working in the software team for the development of the module is equal to or less than 04.
Large Module - The number of persons working in the software team for development of module is greater than 04.

4 - What is the maximum no of persons working in the software team for development of a large module -

5 - Who is the head of the Module Team -
I - There is no formal head of the team and the project {P.L } leader take the final decision about the team and is the only head of the team.
II - There is a Module leader { M.L } in the team.
III - Any other , please specify.

Type of Software	Small Module	Large Module
Application Software		
System Software		

* Please fill the appropriate answers in the boxes given below

6 - If the option II in Question 5 is a choice. Then, what is the educational qualification, expertise and no of year of experience required for M.L?

Type of Software	Small Module	Large Module
Application Software		
System Software		

7 - Who Designs the structure of the module and develops the algorithm for its development ?

I - Only Project Leader.

II - Only Module Leader.

III - Project leader in consultation with the Module Leader.

IV - Project leader in consultation with the team members.

V - Any other , please specify.

Type of Software	Small Module	Large Module
Application Software		
System Software		

8 - Who ensures the co-ordination between various module teams working for the same project and between developer and user of software ?

I - Project leader ensures that.

II - The various module leaders ensures it by helding meetings between them from time to time which is presided by the project leader. If this is the case , what is the frequency of meeting in a month in the design phase.

III - Every member of the teams participates in the meeting held between the member of the project.

IV - Any other , please specify.

Type of Software	Small Module	Large Module
Application Software		
System Software		

9 - Who do the administrative work like allocation of resources , getting money from organization, addition of manpower in the middle of the development work, etc. ?

I -Module leader in consultation with a administrator who is a team member.

II - Project leader only in consultation with top management.

III - Project leader in consultation with the Module leader

IV - Any other , please specify

Type of Software	Small Module	Large Module
Application Software		
System Software		

10 - If the project leader does the design work of the module in consultation with the module leader, is there a person specifically assigned in the team to help him . If yes then what is the educational qualification, expertise, and no of year of experience required for him.

Type of Software	Small Module	Large Module
Application Software		
System Software		

11 - How much all other team members are involved in the design process, give number from 0 to 6

0 - Only M.L is involved , 1- Only M.L is involved but other team member remain present in the meting

2 - Only one member except M.L

3 - Less than half members , 4 - Nearly half of members involved

5 - More than half members, 6 -All members are very much involved.

Please specify the number.

Type of Software	Small Module	Large Module
Application Software		
System Software		

12 - How the difficult technical aspects of the application is taken care ?

I - Project Leader is expected to know all about the application.

II - M.L is expected to know a lot about the application.

III - A member of the team is a specialist in the application.

IV - The help of a specialist is taken during requirement.

IV - Any other , please specify.

Type of Software	Small Module	large Module
Application Software		
System Software		

13 -. Who edit, reviews and criticizes the design by Project leader and Module leader? If required give separate answers for both part of the question.

I - A member of Team assigned that job (Editor).

II - An outside member from other team reviews it.

III - It is discussed in a Brain Storming session of the team.

IV - If any other , please specify.

Type of Software	Small Module	Large Module
Application Software		
System Software		

14 - (I) Does the editor of team have to do coding work ? **Write yes or no.**
(II) What is the qualification requird for him ?

Type of Software	Small Module	Large Module
Application Software		
System Software		

15 - Who solves the problems associated with the Hardware & Software tools required by the team in the development work.

I - A person assigned this specific job in the team.
II - A person having good knowledge serves different teams.
III - All team members are expected to have a good knowledge of this and a more knowledgeable person solves the difficult problems.
IV - Any other , please specify.

Type of Software	Small Module	Large Module
Application Software		
System Software		

16 - Who tests the functioning of the Software Module developed by the team ?

I - Any Team Member
II - Outsider belonging to other module team
III - There is no module level testing and it is done only at project level
IV - Any other , please specify in the box.

Type of Software	Small Module	Large Module
Application Software		
System Software		

17 - Who ensures the quality of the Software Module

I - A person appointed by quality department and a member of team

II - Member of other team

III - The quality department checks the quality of module.

IV - Any other , please specify

Type of Software	Small Module	Large Module
Application Software		
System Software		

18 - Are there some persons in the team , who are concerned with only coding part of software , without bothering about the design aspect, **write yes or no**. If yes , then what is the educational qualification and no of year of experience required for them ? Do they take any other responsibility other than coding work ? If yes , please specify them.

Type of Software	Small Module	Large Module
Application Software		
System Software		

19 - Is any person in the team is provided with secretary ?
If yes , who are they ?

Type of Software	Small Module	Large Module
Application Software		
System Software		

20 - How the documents, previously written programms related to software module is made and maintained inside the team, which can be accessed by any person at the time of need.

I - A Programming Clerk is appointed in the team for this purpose.
II - This is a part time job for the programmer.
III - The secretaries take this additional job.
IV - Any other , please specify.

Type of Software	Small Module	Large Module
Application Software		
System Software		

21 - How any decision about difficulty in design of software module is taken up in the team, if in implementation stage it gives trouble ?

I - By brain-storming
II - It goes to the Project Leader.
III - M.L takes the decision and pass it to other.
IV - It is discussed in team in brain storming session but M.L gives the final decision.
V - Any other , please specify.

If all the steps I, II, III and IV are involved in the process, then write the order in which it takes place

Type of Software	Small Module	Large Module
Application Software		
System Software		

22 - What is the mode of communication inside the team during the implementation of design i.e. coding work

I - By passing memo , paper , documents etc. .
II - By conducting meetings inside the team.
III - Informal communication (i.e. the members discuss with one another their problems)
IV - Any other , please specify.

Type of Software	Small Module	Large Module
Application Software		
System Software		

23 - How many meetings are held in a month between team members during the design stage and during implementation of design phase ? Please give separate answers for both.

Type of Software	Small Module	Large Module
Application Software		
System Software		

24 - Who facilitates the meetings of team members ?

I - A person from the team is assigned this job .

II - Ensured by M.L

III - Any other , please specify.

Type of Software	Small Module	Large Module
Application Software		
System Software		

25- Who keeps track of the issues discussed in the meetings and notes it down in a systematic manner ?

I - A person from the team is assigned this job .

II - It is a part time work of a coder.

III - The clerk in the team is expected to do this job.

IV - The secretary in the team do this work.

V - Any other , please specify .

Type of Software	Small Module	Large Module
Application Software		
System Software		

26 -Does the P.L or M.L who is a formal leader of team (I) write some code and (II) test each line of code written by other programmer ?.(III) Do the code written by one of the programmer is checked by other programmer before submission to program librarian ? **Please write yer or no** for all the questions

Type of Software	Small Module			Large Module		
Application Software	(I)	(II)	(III)	(I)	(II)	(III)
System Software	(I)	(II)	(III)	(I)	(II)	(III)

27 - Does the same team in the company do the maintenance and development work ? If yes , then generally how much time is allocated for maintenance work. If No, then select from the options.

I - The maintenance team is consists of more experienced programmer and headed by same module leader as development team.

II - It works separately under the P.L of development team.

III - Any other, please specify.

Type of Software	Small Module	Large Module
Application Software		
System Software		

28 - Who decides the difficulties in the maintenance , so that these difficulties will tried to be removed in the development phase ?

I - M.L

II - M.L with help of one team member .

III - A consensus among the team members .

IV - Any other , please specify .

Type of Software	Small Module	Large Module
Application Software		
System Software		

29 - If separate teams are there for development and maintenance work. How do the maintenance people participate in the development process i.e. during the selection of algorithm and editing of the software?

I - The leader of maintenance team gives his opinion during design stage to P.L or M.L whoever designs the structure of software

II - The maintenance team has a equal stake in the design.

III - There is a exchange of programmer between development and maintenance team to facilitate co-ordination between two teams.

IV - Any other , please specify.

Type of Software	Small Module	Large Module
Application Software		
System Software		

Any other comments you would like to include -

Thank You for filling this questionnaire.

7.5 *APPENDIX 5*

7.5.1 *Characteristics of Structured Open Team*

Co-ordination : The Co-ordination in structured open team is based on adaptive collaboration i.e. integrating innovation with stability and individual with collective interests through negotiation and discussion. The role and responsibility of the members are flexibly shared. The critical success factor is likely to be interpersonal skills.

System Regulation : The system regulation of this type of team is combined feedback and flexible responsiveness. All the members are encouraged to give their feedback in every issue through meetings. The decision is reached through negotiated, consensus by group process.

Priorities : The priorities of the team are stability and change, group and individual, adaptive effectiveness. The group priorities lies in obtaining balance in all the priorities. The group aims at obtaining stability and be responsive to change of process. The role of each individual should be defined properly keeping in view of the group goal.

Strong Suit : The strong suit of structured open team is adaptation to practical situations of problem solving. The members share information freely to achieve team goal.

Weak Areas : The efficiency of this type of team is low due to waste of time in consensus and negotiation. It is not suitable for smooth, simple and repetitive operation.

Best Application : The best application area is complex problem solving. It is best suitable for application like software development.

Failure Mode : The failure mode of this type of team is chaotic enmeshment and endless processing. The situation may reach a point where everybody is talking and nobody is listening without reaching any conclusion of discussed issue.

Objectives : The objective of the team should be in systematic planning, strategy making, agenda setting, skill building and clarifying the roles of the members.

Form and Style : The form and style of the team should be cooperative, explorative, strategic flexible, learning and practice

Nachmias, C. & Nachmias, D. 1981. Research Methods in the Social Sciences. London: Edward Arnold.

Kraft, P. 1977. Programmers and Managers: The Routinization of Computer Programming in the United States. New York: Springer-Veriag.

Heeks, R. 1996. India's Software Industry: State Policy, Liberalisation and Industrial Development. New Delhi: Sage Publications.

Brooks, B. 1982. The Mythical Man-Month: Eassys on Software Development. Ontario: Addison-Wesley Publishing Company.

McClure, C. 1981. Managing Software Development and Maintenance. New York: Van Nostrand Reinhold Company.

Gunther, R.C. 1978. Management Methodology for Software Product Engineering. New York: Wiley Interscience Publication.

Pressman, R.S. 1988. Software Engineering: A Practitioner's Approach. New Delhi: McGraw-Hill International Editions.

Seigel, S. 1956. Nonparametric Statistics for the Behavioral Sciences. New York: McGraw-Hill Book Company Inc.

Weinberg, G. 1972. The Psychology of Computer Programming. New Delhi: McGraw-Hill Book Company Inc.

Dunn, R.H & Ullman, R.S. 1994. TQM for Compter Software. New Delhi: McGraw-Hill Inc.

Scacchi, W. 1984. Managing Software Engineering Projects: A Social Analysis. IEEE Transactions on Software Engineering, Jan: 49-67.

Schware, Roberts. 1992. Software Industry Entry Strategies for Developing Countries: A 'Walking on Two Legs' Proposition. World Development, 20, No.2: 143 - 164.

Swanson, C.B. & Beath, C.M. 1990. Departmentalisation in Software Development and Maintenance. Communication of the ACM, Vol 33(6): 66 - 77.

Basili, V.R.& Caldiera, G. 1995. Improving Software Quality by Reusing Knowledge and Experience. Sloan Management Review, 21: 55-64.

Rettig, M. 1990. Software Teams. Communication of the ACM, 33(10): 23-27.

Constantine, L.L & Lockwood, L.A.D. 1993. Orchestrating Project Organisation and Management. **Communication of the ACM,** 36(10): 31-45.

Rettig, M & Simons, G. 1993. A Project Planning and Development Process for Small Teams. **Communication of the ACM,** 36(10): 46-54.

Hyman, R.B. 1993. Creative Chaos in High Performance Team - An Experience Report. **Communication of the ACM,** 36(10): 56-62.

Walz, D.B., Elam, J.J & Curtis, B. 1993. Inside a Software Design Team : Knowledge Acquisition, Sharing and Integration. **Communication of the ACM,** 36(10): 63-75.

Abdel-Hamid, T.K. & Madnick S.E. 1990. The Elusive Silver Lining: How We Fail to Learn from Software Development Failures. **Sloan Management Review,** Fall : 39-48.

Brooks, F. 1995. The Mythical Man-Month: AFTER 20 YEARS. IEEE Software, September: 57-61.

Senge, P.M. 1990 The Leader's New Work: Building Learning Organisations. **Sloan School of Management,** Fall: 7-24.

NASSCOM REPORT. 1995: The Software Industry in India : A Strategic Review.

NASSCOM REPORT. 1996: The Software Industry in India : A Strategic Review.